UNIVERSITETET I OSLO
Institutt for informatikk

Visualisering av klimadata basert på åpne standarder

Masteroppgave
(60 studiepoeng)

Leif Gunnar Olonkin

1. mai 2007

Forord

Mastegradsoppgaven er skrevet som en del av mastergradsstudiet ved Institutt for informatikk, Universitetet i Oslo. Arbeidet har vært todelt. Først en praktisk del for teste ut programvare og programvareløsninger for å høste erfaring og kunnskap samt å kunne verifisere at ideer og løsningsforslag kan brukes i mitt arbeid. Denne oppgaveteksten utgjør andre halvdel av mitt arbeid. Oppgaven drøfter mine erfaringer samt inneholder drøftinger av fremtidige eller alternative løsninger og behov.
Arbeidet har for det meste blitt utført ved Observasjonsdivisjonen på Meteorologisk institutt hvor jeg til daglig er ansatt. Oppgaven ble utført i tidsperioden våren 2006 til våren 2007.

Jeg vil gjerne takke min arbeidsgiver for all støtte og praktiske tiltak slik at jeg kunne fullføre mastergradsarbeidet.
Det er flere grunner til å takke arbeidsgiver og da i særdeleshet Klimadivisjonen og GIS gruppen for å ha gitt meg en interessant oppgave å jobbe med. Jeg vil spesielt takke Ole Einar Tveito, Solfrid Agersten, Hanna Szewczyk-Bartnicka og Åse Moen Vidal som har supplert meg med faglig innsikt i klimadata samt gitt meg full tilgang til nødvendige ressurser – i det hele tatt vært veldig positive og imøtekommende.
Jeg vil også få takke Sofus Linge Lystad for mange interessante diskusjoner i kveldstimene samt et sjenerøst utlån av relevant litteratur.
Jeg vil også få rette en takk til mine nærmeste kolleger i mitt daglige virke.

Den mest sentrale personen for gjennomføringen av oppgaven har vært min mentor 1. amanuensis Gerhard Skagestein. Jeg vil rette en stor takk til ham for sjenerøs behandling samt konstruktiv kritikk og innspill. Uten hans faglige innsikt og fleksible tilpasning i forhold til møter og tidsbruk, så hadde det blitt ytterst tungt å gjennomføre oppgaven. Gerhard Skagestein går av med pensjon og bruker siste rest med arbeidstimer for at jeg skal kunne fullføre mastergraden.
For denne innsatsen vil jeg uttrykke min store takknemlighet og ønske ham til lykke med pensjonisttilværelsen!

Blindern, Oslo april 2007.

Leif Gunnar Olonkin

Innholdsfortegnelse

1 Innledning

Jeg vil starte med å sitere en kollega om begrepene meteorologi og klimatologi:

Meteorologi *er vitenskapen om atmosfæren og dens prosesser og fysikk (ibid). Ordet*

"meteorologi" er sammensatt av gresk "meteoros" (i luft) og "logos" (lære). Faget meteorologi kan spores tilbake til Antikken da Aristoteles 350 f. Kr. skrev "Meteorologica", den første kjente læreboken i meteorologi.

Klimatologi *er vitenskapen om klima. Den innbefatter kartlegging og studium av de atmosfæriske forhold ved hjelp av statistiske metoder. Klimatologi blir kalt bokholderdelen av meteorologi(pers. oversettelse, Lamb 1995, s. 11)* [Lundestad, 2004].

Klimadivisjonen (KD) har som målsetting i å utvikle geografiske informasjonssystemer (GIS) basert på klimadata. Jeg siterer her den delen av virksomhetsplanen for 2005 som omhandler GIS (internt dokument):

Klimadivisjonen (KD) ved Meteorologisk institutt (MI) har som hovedoppgave å ivareta instituttets forpliktelser når det gjelder "å studere Norges klima og gi klimatologiske utredninger".

Gridding av klimadata er et satsingsområde i KD. Geografiske informasjonssystemer (GIS) blir benyttet til å vurdere ulike metoder for analyse og presentasjon av klimatologisk informasjon. Det tas sikte på å utnytte topografiske og andre fysiografiske parametere for å beskrive romlig fordeling av ulike klimaelementer. Metodikk for å beregne griddede månedsdata for Norge er utviklet og tatt i bruk i produksjon av klimatologisk månedsoversikt. Det er utarbeidet en første versjon av et slikt datasett med romlig oppløsning 1x1 km2 for perioden 1901–2005. Det er også utarbeidet en pilotapplikasjon for å lage "sjeldenhetskart" (prosentilkart) basert på denne informasjonen. Metodikk for gridding av døgndata (nedbør) er testet og validert. Her tas det sikte på å bruke informasjon om atmosfærens tilstand (sirkulasjon/stabilitet) ved å bruke data fra værvarslingsmodeller eller reanalyser. En rekke klimatologiske kart er tilrettelagt på digital form.

Klimadivisjonens hovedoppgaver har til nå vært å:

- *motta klimadata fra stasjonsnettet, kvalitetskontrollere og lagre disse*
- *gi innspill om fremtidige behov for klimadata*
- *tilrettelegge klimadata for interne og eksterne brukere*
- *formidle klimadata og tolkninger av dem til allmennheten og til ikke-kommersielle brukere*
- *publisere og videreutvikle ulike typer klimastatistikk*
- *utføre detaljert romlig analyse av ulike klimaelement ("griddede klimakart")*
- *forske på fortidens og fremtidens klima*
- *delta i internasjonalt klimatologisk samarbeid*

Gridding (eng. **grid** = rute) er beregning av rutenettdata. En kan forestille seg et regulært rutenett som dekker Norge. I et regulært rutenett er alle rutene like store. Størrelsen på rutene i et slikt rutenett kan variere fra rutenett til rutenett, men er kjent når rutenettet er definert. Stasjonsnettet i Norge er ikke på noen måte regulært, dvs. avstanden mellom målestasjonene og densiteten av stasjoner varierer regionalt. Gridding er en metode for å beregne klimadata der hvor man ikke har stasjoner, ved å interpolere statistiske målestasjonsdata. Rutene i et rutenett kan variere. En metode kan innebære at rutene tilpasses slik at en kun en stasjon befinner seg i en rute til enhver tid. På den måten vil noen ruter representere reelle tidsserier med målinger. Rutene mellom stasjonsrutene representerer interpolerte nedbørdata. I denne konteksten er det slik man skal forstå begrepet gridding. Interpolasjon av klimadata tar hensyn til terrenghøyden som ruten representerer. Det skulle da være innlysende at vi pga av variasjoner i terrenget ikke ønsker at rutene skal være altfor store.

Det overordnede målet er tenkt realisert gjennom etableringen av et eget GIS prosjekt som involverer en rekke ansatte ved MI.

Oppgavens fokus er å lage et vevbasert rammeverk eller et verktøy for utviklere som ønsker lage interaktive vevbaserte løsninger for visualisering av temporale klimadata. Selv om det spesifikt ikke er uttalt, så har Klimadivisjonen en bestemt mening om hvordan visualisering av klimadata uformingsmessig bør se ut. Om ikke satt på trykk eksisterer det en bestemt konvensjonell metode. Alt man trenger å gjøre er å se på tidligere publikasjoner både, elektronisk og på papir, for å forstå at koroplet kartobjekter[1] er i utstrakt bruk for å visualisere klimadata. Denne tradisjonen er imidlertid å oppfatte som et minimumskrav og ikke et tak for hva som skal være mulig å få til innenfor rammeverket.
Å jobbe med GIS er veldig utfordrende når vi snakker om visuelle løsninger. Å visualisere et tematisk kartobjekt i en vevside uten å formulere krav til vevsideutforming, kartobjektets proporsjoner, tegnforklaringer og informativ tekst er en feil.

Instituttet benytter i dag ESRI GIS programvare. ESRI er et ledende firma innen GIS verktøy med hovedkvarter i Redlands, California (se http://www.esri.com).
Universitetet i Oslo benytter seg også av ESRI verktøy til undervisningsformål, samt deler ut brukerlisenser til hjemme-PC etter behov som et ledd i GIS studiet.
Esri's hovedprodukt er ArcGis Desktop, som er en samling programvare som behandler og visualiserer GIS data.
ESRI sitt utviklerverktøy har relativt høye lisenskostnader. Dette hemmer introduksjonen av GIS på instituttet.

[1] Ordet koroplet stammer fra de to greske ordene choros (areal) og plethos (mengde).

Et koropletkart viser den geografiske fordelingen av kvantitative forekomster ved hjelp av den visuelle variable farger i arealutforming [Bjørke, 2005].

1.1 Innholdsbeskrivelse

I dette kapittelet vil jeg gjøre nærmere rede for oppgavens problemstilling, datagrunnlag og litt om den praktiske gjennomføringen. Jeg prøver meg avslutningsvis på en grovskisse av rammeverket.

I kapittel 2 går beskriver jeg datagrunnlaget. Hva er nedbør? Vi beskriver temporale aspekter ved statistiske nedbørdata som er brukt i visualiseringer utover i oppgaven.

I kapittel 3 begynner en rammestruktur å ta form. Vi ser på åpen kildekode som er tilgjengelig samt relevante standarder og vi gjør de valg som skal til og som oppfyller kravene vi har til rammeverket.

I kapittel 4 gis en basal innføring i kartografi. Jeg prøver her å gi forklaring på de viktigste termer i kartografi man må kjenne til, ved bruk av GIS-verktøy.

Kapittel 5 dreier seg om grunnleggende geometri. Vi ser på måter å modellere nedbørdata. Stikkord er 2D flater, punkter og linjer modeller av nedbørdata.

Kapittel 6 tar for seg det som vel utgjør hjertet i rammeverket, nemlig GRASS (Geographic Resources Analysis Support System). Stikkord for dette kapittelet er analyse, transformasjon med mer.

Kapittel 7 går dypere inn i materien til et annet viktig valg nemlig PostgreSQL/PostGIS. Stikkordet her er geospatial database.

I kapittel 8 drøftes visualisering og grafikk standardene SVG og X3D samt standarden for geografisk informasjon GML. I dette kapitlet vil jeg begrunne valget av SVG.

Avslutningsvis i kapittel 9 vil jeg konkludere med hvilke erfaringer jeg har gjort i denne oppgaven, og jeg vil synse litt om fremtiden med bruk av åpne standarder og alternativ bruk av rammeverket.

1.2 Problemstilling

Masteroppgavens problemstilling er å utvikle en bruker- og utviklervennlig vevbasert GIS løsning for visualisering av temporale og spatiale klimadata i 2D og 3D. Løsningsforslaget skal være basert på åpen kildekode og åpne standarder.

Valget av åpne standarder og åpen kildekode støtter overordnet strategisk programvarepolitikk [Hafskjold, 2004]. Overordnet programvarepolitikk oversettes i denne konteksten til å bety ”å tilby et brukergrensesnitt for GIS data for interne MI brukere og allmennheten (alle brukere tilknyttet Internett).
Ser man bort fra regionale og nasjonale målsettinger, så kan man argumentere for at bruk av åpne standarder er hensiktsmessig sett ut fra MI sin målsetting om å videreutvikle og forbedre teknologiske løsninger for internasjonalt samarbeid. En form for internasjonalt samarbeid er utveksling av klimadata.

I forbindelse med etablering av rammeverket er det viktig å definere et par roller i tilknytning til rammeverket, nemlig bruker og utvikler(e).

Utvikler er personer som etablerer og vedlikeholder rammeverket.
Typisk vil det være en systemutvikler i informasjonsteknologi som samarbeider med andre fagpersoner innen geofysikk og matematikk for utvikling av kartdata, som har en klimatologifaglig forankring. En utvikler er ansvarlig for å utvikle tjenester og brukergrensesnitt for datautvelgelse og visualisering av romlig og temporale klimadata.

En bruker er en hvilket som helst person som har interesse av å få laste ned og visualisert klimadata, som for eksempel nedbørdata.
En bruker kan være en fagperson (IT, geofysiker, matematiker, student etc.) som kan laste ned klimadata som et datagrunnlag for vitenskapelig arbeid og visualiseringer. I tillegg har vi den "gjengse" Internettbruker som har en interesse av klimatologi eller som har en eller annen agenda for bruk av klimadata.

Visualisering av klimadata skal realiseres med bruk av vektorgrafikk basert på SVG (Scalable Vector Graphics), som er en de facto W3C (World Wide Web Consortium) standard for 2D grafikk [http://www.w3.org/Graphics/SVG/].
Jeg vil i tillegg også kort drøfte visualisering av klimadata basert på 3D standarden X3D (Extensible 3D). X3D standarden er utarbeidet og vedlikeholdt av Web3D Consortium [http://www.web3d.org/x3d/specifications/].

Rammeverket skal foruten om visualisering også støtte metoder for nedlasting av romlig- og temporale klimadata på XML.
I mitt arbeid har jeg forsket på mulighetene som ligger i bruk av tjenesten WFS (Web Feature Service) og den åpne standarden GML (Geographic Markup Language).

Det er naturlig å splitte problemstillingen opp i to og snakke om en klient- og en tjener løsning. MI vil typisk stå for vedlikehold og utvikling av tjenerløsningen for å tilby geodata i form av kartobjekter eller geodata til brukeren. Klienten kan være en intern arbeidsstasjon på MI eller hos en annen etat (samarbeidspartner) eller i hjemmet.

Jeg betegner tjenerløsningen i mitt rammeverk som en kartdatatjener eller GIS tjener. Det kan sees på som litt snevert og kanskje til dels feilaktigbetegnelse eller gi feilaktige assosiasjoner.

Hvorfor ikke kalle tjeneren i rammeverket for klimadatatjener?

Det ville ikke vært galt, men det er også faktisk en forringet beskrivelse av mulighetene som ligger i rammeverket. Jeg vil begrunne valget ved å si at rammeverket er ikke kun egnet til å behandle klimadata. Alle mulige geospatiale data skal i prinsippet kunne visualiseres eller lastes ned via rammeverket. GIS tjeneren skal i prinsippet kunne binde sammen alle typer georefererte data. Produksjon av klimadata krever rutiner som er utarbeidet på faglig grunnlag.
Det ville bidratt til å se hvilke muligheter som ligger i rammeverket, men jeg har valgt å drøfte dette ut fra et fremtidsperspektiv på slutten av oppgaven. Vi kommer inn på interpolering av data i kapitelene som omhandler GRASS og geometrisk modellering.

Ut fra rollebeskrivelsen tidligere så er det to måter å interagere med rammeverket på.

Utvikler vil typisk benytte manuelle interaktive metoder med bruk av GIS komponenter i systemet for å utvikle ”on the fly” rutiner for visualisering hos klienten og brukeren. Interaktiv bruk vil typisk innebære bearbeiding av nedbørdata (siling, formatering, kartalgebra) og terrengmodellering.
Vi kan kalle denne fasen for kartdatautvikling. Interaktiv bruk vil senere avløses av automatiserte rutiner for å gjøre arkitekturen dynamisk.
Behandling av nedbørdata utover datagrunnlaget vil måtte mellomlagres i rammeverket. Nedbørdata kan også fordelaktig lagres på mer permanent basis for gjenbruk. Rammeverket må derfor kunne håndtere lagring av nedbørdata. Generelt må nedbørdata kunne lagres både på raster- og vektorformat. Vi skal senere se at analyse av geodata baseres seg på at rammeverket kan håndtere lagring av både raster- og vektorstruktur.

Karttjenerløsningen må inneholde metoder for formatering av kartdatadokumenter som skal presenteres på skjerm eller plotter. Rent teknologisk kan dette arbeidet pålegges både tjener og klienten. Man ønsker å sikre riktig formatering i henhold til standard – feil formatering korrigeres kun en plass. Dette skal bidra til å oppnå bedre brukervennlighet. Ved å bruke anbefalte klientapplikasjoner installert på klienten skal brukeren få tilgang til interaktive brukergrensesnitt. Utvikler sin rolle er primært å administrere og videreutvikle GIS tjeneren for å sikre konforme løsninger.

Kartdatatjeneren er tenkt utviklet med henblikk på enkelt vedlikehold og fleksibel og dynamisk eksport av nedbørdata til klienten.

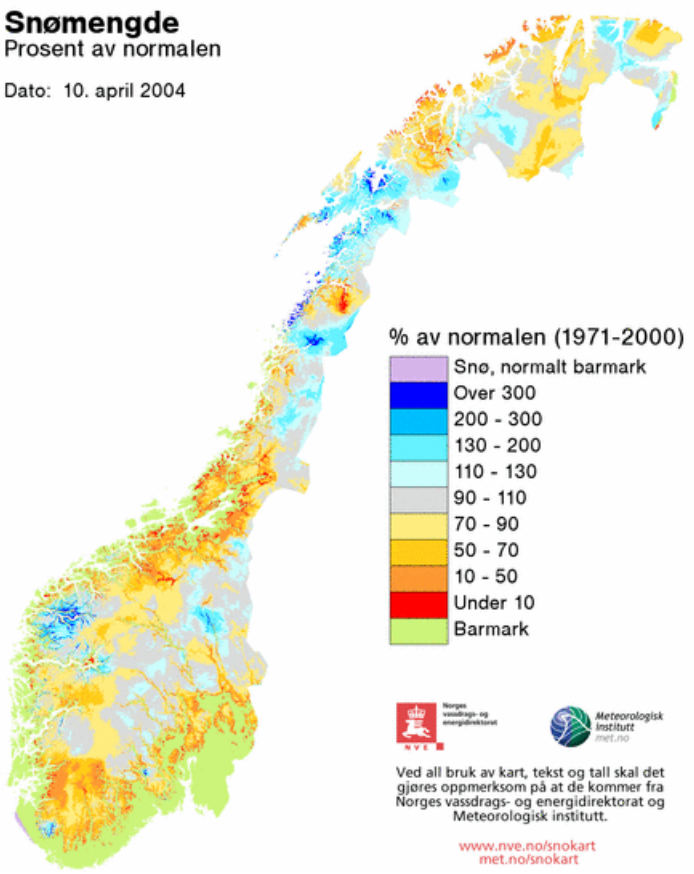
Klientapplikasjonen vil være en nettleser. Nettleseren må støtte, direkte eller indirekte, kartdatadokumenter på SVG format. Nettleseren må være lett tilgjengelig for brukeren for installasjon på klienten. Nettleseren bør kunne kjøre på flere plattformer og være basert på åpen kildekode. SVG er en åpen standard som igjen støtter andre åpne standarder innbygget i et SVG dokument. Det setter krav til klientapplikasjonen om å støtte andre relevante, åpne standarder i forbindelse med utveksling og visualisering av kartdata basert på geodata. Brukergrensesnittet er typisk todelt. En del står for visualisering, mens den andre delen er det vi kaller klientprosesseringsdelen. En typisk klientprosess er fortolkning av SVG dokumenter.

I mitt arbeid har jeg valgt et brukergrensesnitt basert på nettlesere som er åpen kildekode (Mozilla Firefox) eller gratis programvare (Opera).
Hensikten med å benytte nettleser er å unngå behovet for spesialprogrammer. Bruker skal slippe dyre investeringer i programvare, samt unngå begrensinger som følge av valgt klientplattform.
For klienten skal det være valgfrihet ved interaktiv bruk i størst mulig grad, men like viktig er å gi klienten tilgang til selve kartdata.

Figur 1 (snøakkumuleringskart) illustrerer hvordan man tenker seg nedbørdata visualisert. Dette er et konvensjonelt koroplet kartdokument på rasterformatet JPEG. Utseendemessig vil dette dokumentet likne et kartobjekt i en visualisering formatert via mitt rammeverk.
Figuren illustrerer mengden akkumulert nedbør i form av snø i predefinerte intervaller og er fargekodet. Kartdokumentet gir to typer informasjon, nemlig hvor stor nedbørmengde og hvor nedbøren har falt.
Slike kartdata har til nå vært produsert med interaktiv bruk av et GIS verktøy. Resultatet er så blitt eksportert til et rasterformat som man har gjort tilgjengelig via en statisk vevside.

Valget av denne type format skyldes at enkelte prosesser er veldig tidkrevende. Da tenker jeg spesielt på produksjonen av nedbørdata, som er interpolerte nedbørdata fra målestasjoner, for å gi et bilde av snømengden over hele Norge.
Kartdokumentet er tilgjengelig for nettlesere, men vevsiden er på ingen måte interoperabel. Der er ingen dynamiske verktøy som zoom knapper eller interoperable temporale verktøy for å velge andre tidsserier kunne gi brukeren visuelle perspektivvalg for visualisering av nedbørdata.
I skrivende stund har det kommet til flere løsninger som gir et mer dynamisk innhold til vervsidene, men fortsatt formaterer man kartdokumenter på rasterformat.

Figur 1: JPEG snøakkumuleringskart. Se også http://met.no/snokart/2004/2004_04_10/swepr.shtml



Hovedmålet i oppgaven er å visualisere interpolerte nedbørdata. Oppdragsgiver har hatt bestemte meninger om av hva slags verktøy som inntil videre skal benyttes for interpolering av nedbørdata.
Implisitt skal dette tolkes som et ønske om kvalitetssikring av nedbørdata via kunnskap og forskning realisert gjennom et verktøy hvor man har opparbeidet seg brukerkompetanse. Dette er en typisk problemstilling i forbindelse med å utvikle integrerte løsninger. Det tar tid å bytte verktøy, endre rutiner samt verifisere at kvalitet og fleksibilitet er ivaretatt.

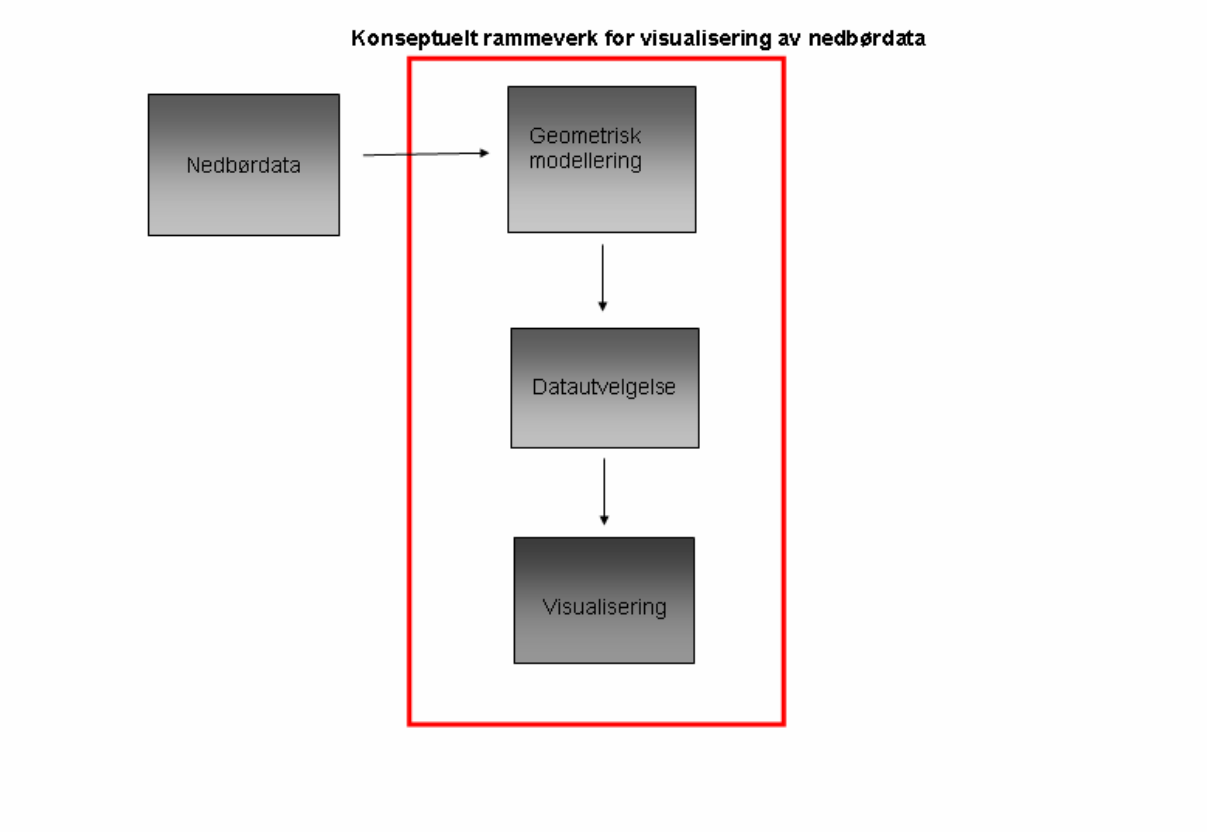
Interpolering av nedbørdata er gjenstand for forskning og endringer i rutiner. Det er også en CPU intensiv affære. Det er fullt ut mulig å integrere interpolering inn i mitt arbeide, men ut fra min tidligere argumentasjon om å skille ut datagrunnlaget og se på det som en ekstern og faglig kildedatatjener har jeg utelatt produksjon av datagrunnlaget som en del av mitt

rammeverk. Jeg vil vise at mitt rammeverk støtter interpolering av nedbørdata i kapittel 6 som omhandler GRASS.

Jeg har ikke fokusert så mye på visuelle løsninger men mer på utvikle rammeverk som formaterer kartdokumenter som er dynamiske og interoperable. Visuelle løsninger er viktig for meg på den måten at de skal gi så korrekt tematisk klimainformasjon om nedbørdata som mulig. Jeg har utarbeidet eksempler på hvordan dette kan gjøres, men der er helt sikkert mange alternative og mer attraktive visuelle løsninger.

Jeg avslutter dette avsnittet med Figur 2 som illustrerer konseptuelt rammeverk og som er utgangspunkt for videre drøftelser. Rammeverket som er drøftet er innenfor den røde rammen, og nedbørdata er inndata til rammeverket.

Figur 2:Konseptuelt rammeverk for vevbasert visualisering av nedbørdata



1.3 Avgrensninger

I min oppgave drøfter jeg et løsningsforslag på rammeverk basert på nedbørdata (snø og regn)[2] som datagrunnlag. Det er et bevisst valg for å gjøre teksten og løsningsforslagene mer sammenhengende. Problematikk som går på grafisk utforming og formatering av vevdokumenter pga iboende egenskaper i kildedata er utelatt i drøftelsen. Forskjellige typer klimaparametere gir forskjellige føringer for hva som er attraktiv og pedagogisk visualisering,

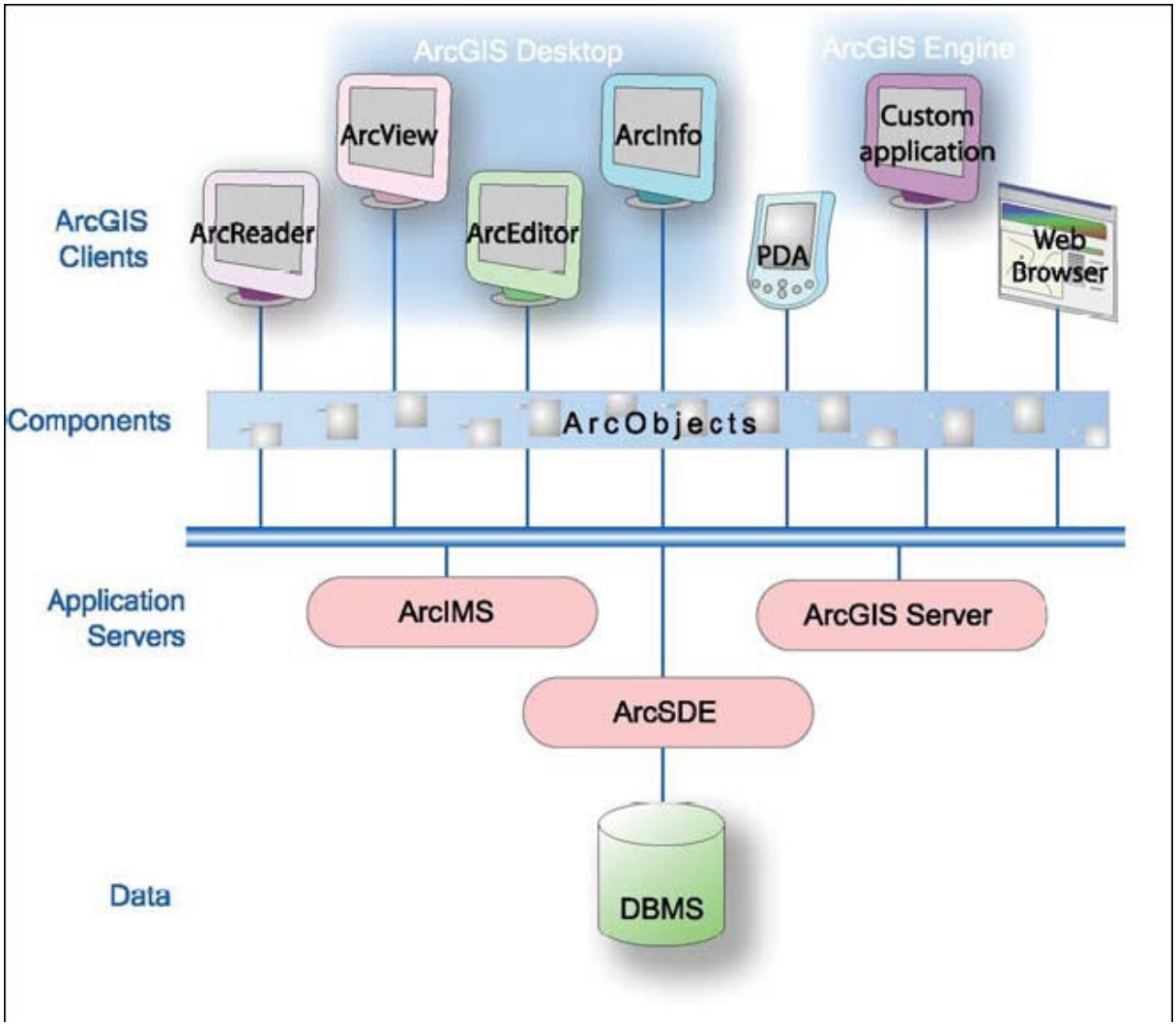
[2] Nedbør måles som kjent i mm. En presisering er at nedbør måles som en nedbørhøyde over et areal på 1x1 $m^{2.}$

og som kan bidra til intuitiv forståelse av topografiens påvirkning på været. Problematikk som går på grafisk utforming medfører ikke strukturelle endringer i rammeverket.

1.4 Alternative løsninger.

Jeg vil kort komme inn på andre GIS løsninger for å anskueliggjøre hva som rører seg i GIS verdenen i dag.
Jeg har allerede nevnt ESRI, som leverer et komplett og proprietært GIS verktøy. ESRI produkter mye benyttet og er verdt å nevne som et eksempel. Se Figur 3.
ArcIMS er et vevutviklerverktøy for å gjøre kartproduktene tilgjengelig via en nettleser. Et slikt verktøy samsvarer med målsettingen til mitt rammeverk Selv om ArcGIS er en proprietær løsning, så er der innbygget støtte for åpne standarder som definert av OGC (Open Geospatial Consortium, se http://www.opengeospatial.org/). OGC er en gruppe av bedrifter og universiteter som arbeider for åpne standarder i utveksling av geografisk informasjon.
Det er interessant å observere at åpne standarder faktisk har innflytelse på utviklingen av kommersielle produkter.

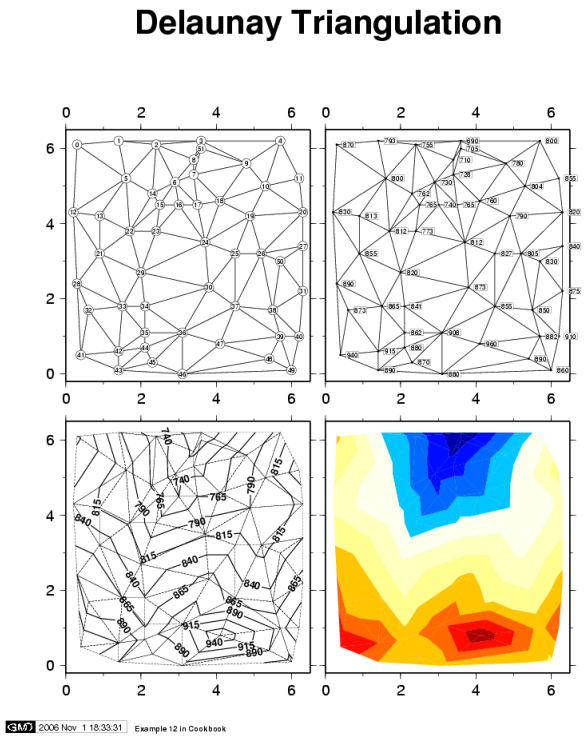
Figur 3: ArcGIS arkitektur
[http://edndoc.esri.com/arcobjects/9.1/ArcGISDesktop/DesktopDevGD_Ch1.pdf]



Figur 3 representerer altså en kommersielle og proprietær løsning. Det som imidlertid er interessant, er å se på løsninger med bruk av åpne standarder og basert på åpen kildekode. Internett stedene Open Source GIS (Se http://opensourcegis.org/) og Free GIS (Se http://freegis.org/) er ypperlige startpunkter for å finne åpen kildekode GIS løsninger. Her finnes blant annet referanser til kode som allerede nevnt i min GIS løsning. De er nevnt i neste avsnitt hvor på enkelte av disse vil bli mer utførlig drøftet i neste kapittel.

GMT (akronym for The Generic Mapping Tools) er utviklet av Paul Wessel og Walter H. F. Smith med hjelp fra frivillige fra hele verden. GMT er tilgengelig under lisensen GNU GPL(General Public License). Se Figur 4 som er et eksempel på kart med Delaunay triangulering.
GMT er en samling på ca. 60 verktøy for å manipulere geodetiske og kartesiske kartsett. Visualiseringen er formatert som EPS (Encapsulated Postscript) rangert fra simple x-y rissinger, konturkart, kunstige belyste flatekart og 3D kart.
GMT har innbygget støtte for over 30 projeksjoner og projeksjonskonverteringer.

Figur 4: GMT eksempel



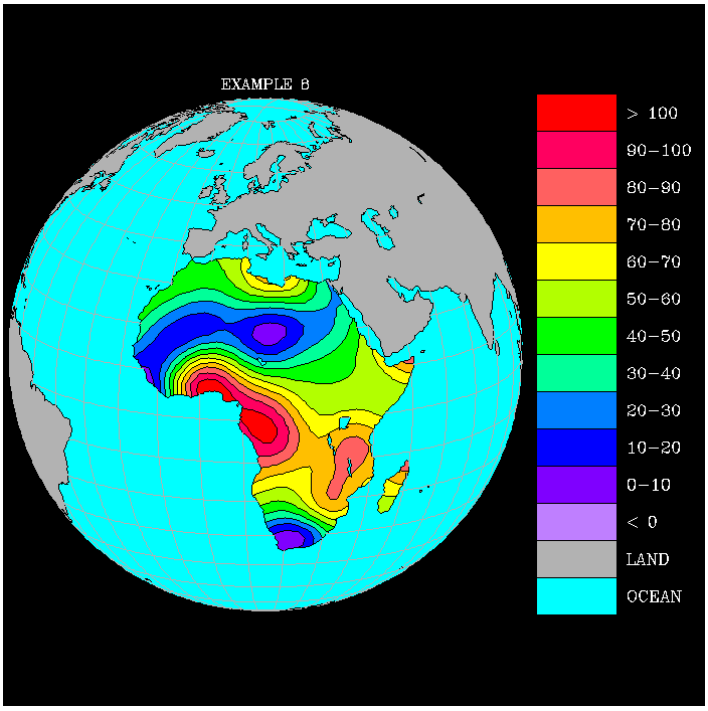
GMT kommer med en utførlig manual og er et komplett GIS verktøy, med funksjoner for dataanalyse, kartproduksjon og visualisering. Se også http://gmt.soest.hawaii.edu/ .

NCAR Graphics (National Center for Atmospheric Research) er åpen kildekode (GNU GPL lisens) hvor man kan laste ned kildekode eller binær kode for installasjon og kjøring. Se også http://www.ncar.ucar.edu/. NCAR Graphics er består av

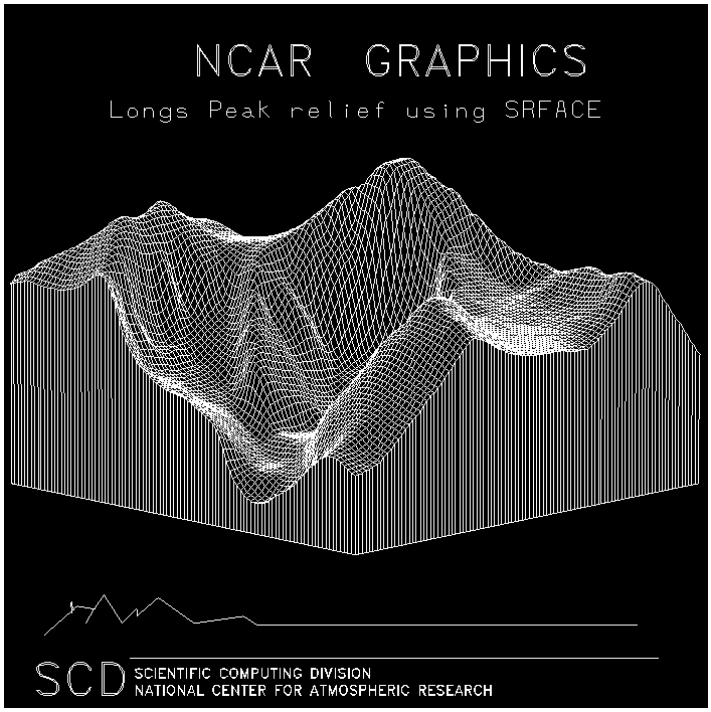
- bibliotek med over to dusin Fortran/C moduler for å visualisere kart, vektorer, "streamlines", værkart, flater, histogrammer, X/Y plot, tekst og tegnforklaringer.
- ANSI/ISO standardversjon av GKS (Graphical Kernel System, se http://wwwasdoc.web.cern.ch/wwwasdoc/WWW/gks/gksprimer.html) med både C og FORTRAN funksjonskall.
- et matematikk bibliotek med en samling C/Fortran funksjoner for 1D, 2D og 3D interpolering eller estimering.
- verktøy for å visualisere, editere og manipulere grafikk.
- kartdatabase.
- hundrevis av FORTRAN og C kode eksempler.
- demo programmer.
- ferdige skripts for kompilering av kode til kundespesifiserte applikasjoner.

NCAR Graphics er på lik linje med GMT et sammensatt og fleksibelt GIS verktøy. Se visuelle eksempler i Figur 5 og Figur 6. Se http://ngwww.ucar.edu/ng4.4/index.html.

Figur 5: NCAR Graphics eksempel 1

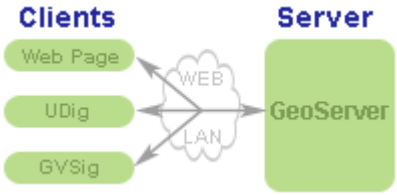


Figur 6: NCAR Graphics eksempel 2

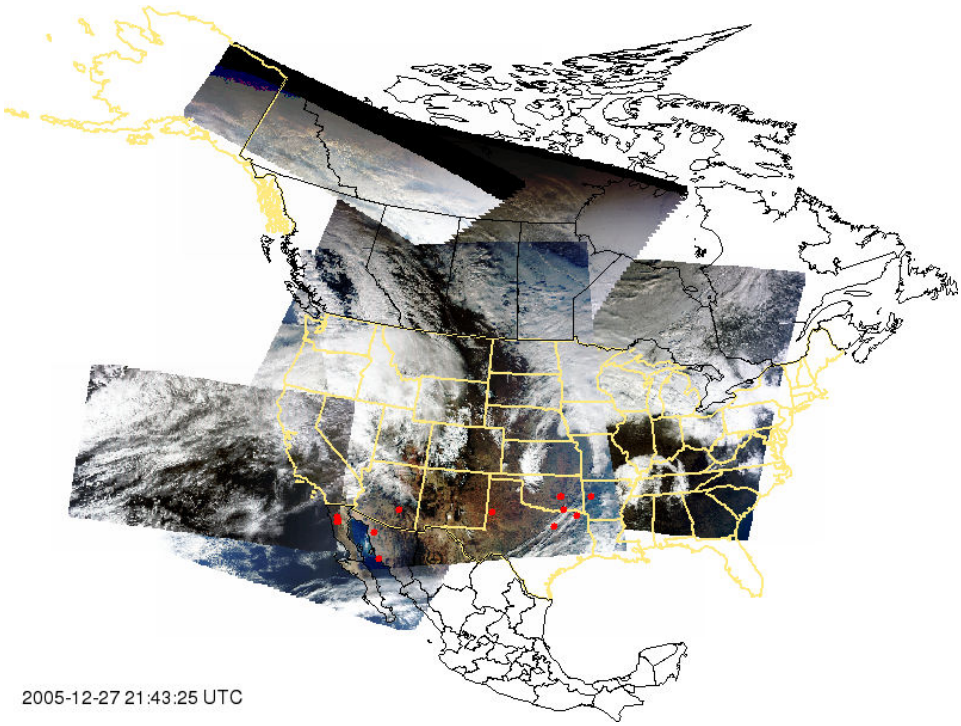


GeoServer er et annet eksempel jeg vil trekke frem. GeoServer er bygget på GeoTools (se http://geotools.codehaus.org/), som er et Java-bibliotek for standardisert manipulering av geospatiale data basert på OGC standarder. GeoTools er et bibliotek for utviklere til å lage egne GIS løsninger, og er åpen kildekode.
GeoServer er utviklet med J2EE (Java 2 Platform, Enterprise Edition fra Sun Microsystems, Inc.) for visualisering og editering av kart og kartdata. GeoServer har innbygget støtte for standardene WFS-T (Web Feature Service-Transactional) og WMS (Web Map Service) protokollene fra OGC til å produsere JPEG (Joint Photographic Experts Group, ISO/CCITT standard), PNG (Portable Network Graphics, ISO/IEC 15948:2003), SVG (Scaleable Vector Graphics, W3C), KML/KMZ (Keyhole Markup Language,KMZ = komprimert KML, merkevare for Google Earth og Google Maps), GML (Geography Markup Language, OGC), PDF (Portable Document Format, utviklet av Adobe Systems), Shapefiles (geografisk vektor filformat, utviklet av ESRI) med mer.
Med GeoServer kan man altså publisere kart/bilder ved å benytte seg av WMS protokollen. GeoServer støtter overføring av geografisk informasjon på GML format med bruk av WFS standarden.
WFS-T er en OGC standard som gjør det mulig å redigere geospatiale data i såkalte transaksjonsblokker, dvs. å utføre operasjoner som sletting, oppdatering eller endring og legge til nye data.
GeoServer sin målsetting er å gi støtte for de nevnte og unevnte standardene og fungere som et bindeverk mellom legacy databaser og forskjellige klienter i det som kalles den geospatiale veven. Den Geospatiale veven er åpen nettløsning for utveksling av geospatiale data, og er således analogt med begrepet Internett.
Se Figur 7 som er en prinsipiell topologi ved bruk av GeoServer, og Figur 8 og Figur 9 som visuelle eksempler produsert med GeoServer.
Se http://docs.codehaus.org/display/GEOS/Home.

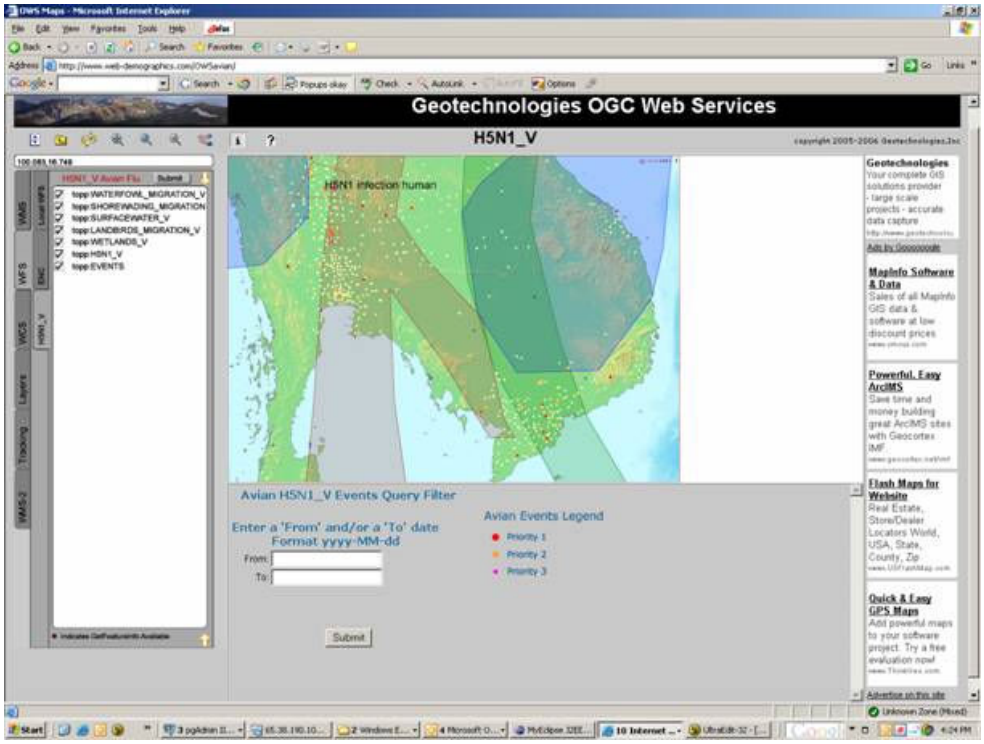
Figur 7: GeoServer



Figur 8: Mosaikk av satellittbilder, rasterformat



Figur 9: Interaktiv GIS visualisering med SVG



Der er mange åpne kildekodeløsninger for mer eller mindre komplekse GIS-løsninger tilgjengelig. Jeg har her drøftet de mer komplekse GIS-løsningene. Spektret av GIS løsninger varierer voldsomt fra enkle geo-romlige editeringsverktøy, verktøy for koordinattransformering, romlige analyseverktøy samt GIS-verktøy med for visualisering. De mer komplekse GIS-løsninger implementerer flere av de nevnte enkeltverktøyene.

1.5 Introduksjon til relevante standarder og programvare

Det finnes mange åpne standarder, åpne kildekodeløsninger samt gratis eller kommersielle standarder som er verdt å kjenne til. Jeg vil her gi en oversikt over de som jeg har vurdert og de som blir benyttet som komponent i rammeverket, samt gi henvisninger og referanse til ytterligere litteratur. Mange av standardene som nevnes her, inngår i mange andre GIS løsninger. I mitt arbeid har jeg konsekvent søkt å benytte meg av åpen kildekode og åpne standarder, som tidligere nevnt i målsettingen. Allikevel er det interessante teknologiske løsninger i den kommersielle verdenen også, og de bør av akademiske årsaker være nevnt. Ofte er dette gratis programvare, eller gratis så lenge man ikke har intensjoner om å benytte programvaren i kommersielt øyemed.

Aller først - hva menes med åpen standard og åpen kildekode?

Jeg har lett i litteratur og på Internett for å se om der finnes en enhetlig forståelse av disse begrepene.
EU har laget utfyllende definisjon om åpen standard som blant annet Teknologirådet - et uavhengig, rådgivende organ som skal vurdere den teknologiske utviklingen på alle samfunnsområder - stiller seg bak [Hafskjold, 2004]:

1. Standarden er anerkjent og vil bli vedlikeholdt av en ikke-kommersiell organisasjon, og det løpende utviklingsarbeidet foregår på basis av beslutningsprosess som er åpen for alle interesserte parter.
2. Standarden er publisert og dokumentasjonen er tilgjengelig, enten gratis eller for en ubetydelig avgift. Det er tillatt for alle å kopiere, distribuere og bruke standarden gratis.
3. Den intellektuelle rettighet knyttet til standarden (f. eks. patenter) er gjort ugjenkallelig tilgjengelig avgiftfritt
4. Det er ingen forbehold om gjenbruk av standarden.

En definisjon på åpen kildekode fra samme kilde:

1. er åpen for innsyn av alle
2. kan modifiseres av alle
3. kan gjenbrukes i annen programvare og
4. kan fritt distribueres til andre

1.5.1 Standarder

- **XML** (eXtensible Markup Language) er et markeringsspråk som benyttes til å beskrive språk som beskriver data. XML er altså et metaspråk. XML er en åpen standard fra W3C. Strukturert informasjon består av ord, setninger, bilder osv.
  XML språk omslutter data med beskrivende "tagger" i det XML baserte dokumentet. XML bruker alltid Unicode tegnsett, for å gi støtte til alle nasjonaliteters språk og skriftsystem. Se også http://www.w3.org/XML/.

- **SOAP** (Simple Object Access Protocol) er en åpen standard fra W3C. SOAP er en enkel protokoll for utveksling av informasjon i et desentralisert, distribuert miljø.
  SOAP sendes i meldinger på samme måte som HTML sider, og inneholder mye likt i meldingshodet. Data sendes som XML i en konvolutt og en hoveddel. I tillegg kan det ligge et XML-skjema som forteller hvordan selve XML-dokumentet skal bygges opp. Se også http://www.w3.org/TR/soap/.

- **WFS** (Web Feature Service) er en åpen OGC standard som beskriver operasjoner for å hente ut kartdata og metadata på GML format. WFS benytter protokollen HTTP (Hypertext Transfer Protocol) hvor data er på XML format og overført med SOAP protokollen. WFS består standardkomponentene for tjenester av WSDL (Web Services Description Language), SOAP og UDDI (Universal Description, Discovery, and Integration). XML må også nevnes som en grunnstein for de tre andre. Forenklet sagt er

WFS et stykke programvare laget for å støtte interoperabel maskin-til-maskin interaksjon over et nettverk med standardiserte nettkomponenter, uavhengig av plattform. Følgende operasjoner er tilgengelig i henhold til standarden:

  - skape nye geometriske objektinstanser
  - slette geometriske objektinstanser
  - oppdatere geometriske objektinstanser
  - låsing av geometriske objektinstanser
  - laste ned eller spørre etter geometriske objektinstanser basert på spatiale eller ikke spatiale kanter

Se også http://www.opengeospatial.org/standards/wfs.

- **WMS** (Web Map Service) er en åpen OGC standard for å produsere skalerbare kart som kan visualiseres. Kartene vil være geografisk refererte i ett eller flere internasjonalt akseptert(e) koordinatsystemer. Denne standarden definerer kart som et portrett av geografisk informasjon i en digital bildefil for å kunne vises på en PC skjerm. Eksempel på bildeformater er PNG, GIF eller JPEG, men også vektorbaserte geografiske elementer i SVG eller WebCGM format. Kartet er presentasjonen på skjermen, og ikke dataene i seg selv. WMS har 3 standard operasjoner:

  - returnerer redigerbare metadata (GetMap).
  - returnerer et kart, der parametere for geografi og utstrekning er definert (GetFeatureInfo).
  - returnerer informasjon om spesielle egenskaper som er vist i kartet (GetCapabilities).

  Se også http://www.opengeospatial.org/standards/wms.

- **GML** (Geography Markup Language) er en åpen standard fra OGC. GML er et markeringsspråk for utveksling av geografisk informasjon. GML er også et XML basert språk. Se også http://www.opengeospatial.org/standards/gml.

- **XSLT** (Extensible Stylesheet Language Transformations) er et XML basert stilarkspråk som beskriver transformasjon av et XML dokument til andre XML dokumenter ved hjelp av XSL. XSLT er en åpen standard fra W3C. Se også http://www.w3.org/TR/xslt.

- **XSL** (Extensible Stylesheet Language) er åpen W3C standard for å definere stilark for og i XML dokumenter. Ved å la XML bli brukt til å representere innhold og XSL for utforming, vil innhold og stil bli separert. Se også http://www.w3.org/Style/XSL/.

- **SVG** (Scaleable Vector Graphics) er et XML basert språk for å beskrive 2D grafikk. SVG er en de facto standard fra W3C. SVG støtter 3 typer grafiske objekter, nemlig tekst, vektorformat (figurer som kan beskrives av punkter, linjer og kurver) og rastergrafikk som for eksempel bilder rasterformat som jpeg, png, bmp etc. Standarden støtter bruk av javaskript og DOM (Document Object Model) samt animasjon. Filtereffekter og skyggelegging er også viktige funksjoner. Formatet er veldig interessant med tanke på bruk innen mobiltelefoni hvor båndbredde er begrenset. I tillegg blir utskrifter av denne

type grafikk pen på printere siden den skalerer. Se http://www.w3.org/Graphics/SVG/. SVG blir drøftet mer inngående i kapittel 8.

- **X3D** (eXtensible 3D) er en ISO (ISO/IEC 14772) standard for å definere interaktiv vev- og distribuert 3D grafikk integrert i multimedia. X3D er en etterfølger til VRML(Virtual Reality Modeling Language) den originale standarden for vevbasert sanntids 3D grafikk. X3D standarden er forenklet sagt en forbedring av VRML på en rekke områder. X3D har en rekke utvidelser i forhold VRML97, som for eksempel Humanoid Animation (H-Anim kilde: http://www.h-anim.org/), NURBS (Non Uniform Rational Basis Spline kilde: http://en.wikipedia.org/wiki/Nonuniform_rational_B-spline) samt GeoVRML (kilde: http://www.ai.sri.com/geovrml/). X3D støtter XML standarden og VRML97 syntaks, og den har en utvidet API for utviklere. Nettlesere støtter X3D gjennom programtillegg, men der er også egne visualiseringsprogrammer for X3D dokumenter. En konsekvens av standarden er at visualiseringsverktøy for X3D også skal kunne tolke VRML97 dokumenter. Se også http://www.web3d.org/x3d/specifications.

- **HTML** (HyperText Markup Language) er et markeringsspråk for laging av nettsider med hypertekst og annen informasjon som kan vises i en nettleser. HTML benyttes til å strukturere informasjon – angi noe tekst som overskrifter, avsnitt, lister etc. Grammatisk struktur er HTML DTD (Document Type Definition) som ble skapt ved å gjøre bruk av metaspråket SGML (Standard Generalized Markup Language ISO 8879:1986) syntaks, som er en internasjonal standard for tekstformatering. HTML ble opprinnelig definert i 1989 av Tim Berners-Lee og Robert Caillau og videreutviklet av IETF (Internet Engineering Task Force) og er nå en internasjonal standard (ISO/IEC 15445:2000). Siden har HTML-spesifikasjonene blitt vedlikeholdt av W3C. Se også http://www.w3.org/MarkUp/.

- **XHTML** (Extensible HyperText Markup Language) er en åpen standard fra W3C basert på XML standarden i stedet for SGML. I all hovedsak er den lik HTML 4, men med strengere syntaktiske regler (lukke alle elementer etc.). Ordet er en sammenføyning av HTML og XML. Grunnen til at man vil ha reglene så strenge, er blant annet at utbredelsen av såkalte WYSIWYG-programmer, som har gjort det lettere å lage internettsider, har ført til mer rot og flere sider som ikke følger standardene. En annen grunn er at sider som følger de rigide, men ryddige reglene blir lettere å lese for små datamaskiner, som for eksempel PDA-er og mobiltelefoner. Se http://www.w3.org/TR/xhtml1/.

- **CSS** (Cascading Style Sheets - på norsk – stilark) har vært en åpen standard i W3C siden 1994, men var ikke fullført før i 1996. CSS2 var klar i 1998. CSS er et språk som brukes til å definere utseende på filer skrevet i HTML eller XML. Prinsippet er at HTML- eller XML dokumentet utelukkende skal beskrive struktur og semantikk, mens utforming, farger og annen stilark informasjon skal beskrives ved hjelp av CSS. Stilark informasjonen kan integreres i selve dokumentet eller skilles ut som en egen fil med endelsen .css. Et ubegrenset antall dokumenter kan referere til samme .css-fil, noe som er styrken i CSS: Ved å endre på en fil, kan man endre fargebruk, bakgrunnsbilder osv. i alle dokumenter som peker til CSS-filen. Se også http://www.w3.org/Style/CSS/. Både XSL og CSS er standarder for stilark. Forskjellen er at CSS kan benyttes i til utforming av

XML og HTML dokumenter, mens XSL er skrevet i XML syntaks ”kun” for utforming av XML dokumenter.

- **ECMAScript** (ECMA akronym for European Computer Manufacturers Association). I 1998 ble JScript og JavaScript overtatt av ECMA og videreført som en felles standard under navnet ECMAScript eller ECMA-262. De fleste kaller det dog fortsatt JavaScript. ECMAScript er en åpen standard. ECMA skript er et objektorientert språk for å foreta beregninger og manipulere programobjekter på en vertsmaskin. Se også http://www.ecma-international.org/publications/standards/Ecma-262.htm.

- **DOM** (Document Object Model) er en åpen standard fra W3C, og beskriver hvordan et programmeringsspråk (f.eks. JavaScript) kan referere og manipulere elementer i et HTML/XML dokument. For å manipulere innholdet/utsende til en vevside etter at siden er vist til brukeren, må DOM benyttes. DOM bestemmer hvilke elementer som kan endres, deres adresse, hvordan de kan endres samt hvordan endringene trigges. Internett Explorer og Netscape hadde i versjon 4 to ulike DOM'er som kalles "Layer DOM" og "All DOM". W3C har i ettertid definert en standard "ID DOM level 1" som nettlesere i mer eller mindre grad har implementert. Selv om JavaScript ofte benyttes, er det viktig å presisere at DOM er et nøytralt API (Application Programming Interface) og kan implementeres i et hvilket som helst programmeringsspråk. Se også http://www.w3.org/DOM/.

- **Adobe Flash** (tidligere Macromedia Flash) eller bare Flash er et proprietært format (ikke basert på åpne standarder) for å publisere animasjoner, multimedia og avanserte applikasjoner på nettet. Flashspilleren er en virtuell kjøretidsmaskin for å tolke Flashfiler. Flashfiler har filutvidelsenavnet fla eller swf (Small Web Format - komprimert .fla) og kan visualiseres i en nettleser via et programtillegg fra Adobe kalt Flashspiller. Termen Flash referer strengt tatt både til Flash IDE (integrated development environment) og Flashspiller. Flash-format er basert på vektorgrafikk, men inkluderer også støtte for rastergrafikk og et skriptspråk kalt ActionScript. Multimedia støttes i form av bidireksjonal dataflyt av lyd og video. Flashfiler kan også eksekveres uten Flashspiller i et kjørbart filformat kalt Projektor. Flash er en populær standard og er ofte satt opp mot SVG for en sammenlikning.

1.5.2 Programvare, bibliotek

- **PostgreSQL** er et objektorientert relasjonsdatabasehåndteringssystem som har sitt opphav fra Berkeley University of California i 1977. Gjennom mange års utvikling har det tjent opp sitt rykte som en pålitelig databaseløsning for flere maskinarkitekturer. Prosjektet legger vekt på å følge SQL standardene ANSI-SQL 92/99. Per dags dato er et subsett implementert, samt egne tillegg til standardene. I så måte skiller PostgreSQL seg ikke fra andre databaseløsninger når det gjelder å følge SQL standarder. PostgreSQL er tilgengelig for mange plattformer som Windows, Linux, Unix og Mac OS for å nevne de mer kjente plattformene. Se også http://www.postgresql.org/.

- **PostGIS** er en utvidelse av PostgreSQL objektrelasjonsdatabase som tillater lagring av GIS objekter. PostGIS benytter spatial indeksering via GiST (Generalized Search Tree),

samt funksjoner for analyse og prosessering av GIS-objekter. GiST er et avansert system som kombinerer en rekke sorterings og søkealgoritmer som inkluderer B-tre, B+-tre, R-tre, samt en rekke andre. Jeg presiserer at GiST er en del av PostgreSQL og ikke en utvidelse som følger med PostGIS. PostGIS er utviklet av Refractions Research som et prosjekt for å utvikle en spatial databaseløsning. PostGIS følger OGC standarden "Simple Features Specification" for SQL. Se også http://postgis.refractions.net/.

- **GRASS** (Geographic Resources Analysis Support System) er en stor samling programmer for statistikk, interpolasjon, arkivering (permanent og mellomlagring), rastering og vektorisering av geodata og kartlaginformasjon. GRASS har innbygde drivere for visualisering av kart og kartlagene som er lagret. I skrivende stund er visualiseringen i GRASS rasterbasert. GRASS kan dog eksportere data både raster- og vektorformat som kan visualiseres av andre programmer. GRASS drøftes mer inngående i et kapittel 6. Se også http://grass.itc.it/.

- **Mozilla Firefox** – er nettleser utviklet av frivillige utviklere i et prosjekt kalt Mozilla. Mozilla Foundation, som er et ”non-profit” selskap lokalisert i California (USA), ble opprettet i 2003 for å sikre at Mozilla prosjektet kunne fortsette, uavhengig av utviklernes rolle i prosjektet. Mozilla Foundation sin rolle er å promotere utvikling og åpen tilgang til kode for nettlesere og andre Internett applikasjoner. Mozilla Firefox er et resultat av selskapets intensjon. Selskapet fikk under oppstarten støtte fra America Online's Netscape divisjon. Nettleseren er ikke bare aktuell fordi den har innbygget støtte for visualisering av SVG dokumenter, men også fordi den er en reell konkurrent til andre nettlesere (både kommersielle og åpen kildekode). Så langt er produktet en suksess, da det er en meget utbredt og populær nettleser. Se også http://www.mozilla.org/foundation/.

- **Opera** (norskutviklet, gratis i 2005) er en nettleser laget for de fleste operativsystemer. Opera støtter alle de vanligst brukte markeringsspråk og skriptspråk som benyttes på Internett, som for eksempel HTML, XML, XHTML, JavaScript og CSS, samt eksperimentell støtte for Web Forms 2.0 som fortsatt er under utviking. Med plugins kan den også tolke Flash, Java samt diverse lyd- og bildepluginer.
  Opera startet som et prosjekt utviklet av Televerket (nå Telenor) i 1992, med da Televerket ikke lenger ønsket å satse på prosjektet gikk Jon Stephenson von Tetzchner og Geir Ivarsøy sammen om å arbeide videre med det i et eget selskap, Opera Software ASA. Kilde - http://no.wikipedia.org/wiki/Opera_nettleser.

- **Perl** (Practical Extraction and Reporting Language) er et skriptspråk/programmeringsspråk utviklet av Larry Wall og ble sluppet desember 1987. Det krever ingen kompilator, men det krever at man har Perl interpreter installert. Perl er optimalisert for enkelt å kunne skanne tekstfiler og skrive ut data i pene rapporter. Det finnes moduler for alle mulige formål. Perl er veldig populært blant brukere og utviklere, og er ofte språket som er benyttet i et CGI (Common Gateway Interface) skript.
  Se også http://www.perl.com/.

- **PHP** (Hypertext-Preprocessor) eller ”Personal Homepage”, som det egentlig het frem til 1995, er et skriptspråk for tjener siden. Koden ligger lagret på tjener og koden eksekveres også på tjeneren. Det kalles preprosessor fordi den er ment til å oppdatere informasjon i forkant av visualiseringen f. eks oppdatere dato, klokke eller annen informasjon innebygd i et vevdokument. PHP er et nyttig verktøy til å lage dynamiske vevsider. HTML alene har ikke muligheten til dette. Med PHP kan du programmere mot databaser, filer, XML,

for å nevne et knippe valg. En stor grunn til at PHP er blitt så populært, er at det er lett å programmere for folk som allerede kan C eller C++. Syntaksen er veldig lik. Se også http://www.php.net/.

- **R** er en statistikkpakke som er utviklet av Ross Ihaka og Robert Gentleman ved University of Auckland, New Zealand. Det er basert på programmeringsspråket S som blir benyttet i en annen statistikkpakke ved navn S-Plus. I dag blir R videreutviklet som R Project av R Development Core Team under The R Foundation. R er fritt tilgjengelig under GNU General Public License. Den er kompatibel med Macintosh, Windows og de fleste Unix-variantene. R er en sammensatt programpakke med fasiliteter for datamanipulering, beregninger og grafisk visualisering. R inkluderer

    - fasiliteter for effektiv datahåndtering og lagring,
    - et sett av operatorer for beregninger på arrays og i særdeleshet matriser,
    - et stort, koherent og integrert samling av mellomliggende verktøy for dataanalyse,
    - grafisk fasiliteter for dataanalyse og visualisering på skjerm eller utskrift, og
    - et godt utviklet, enkel og effektiv programmeringsspråk som inkluderer betingelser, sløyfer, brukerdefinerte rekursive funksjoner samt inn- og ut operasjoner.

  R er utviklet med et standard programmeringsspråk, noe som gjør det mulig for brukere å utvide funksjonaliteten ved å definere egne funksjoner. Mye av språket er selv skrevet i R. For intensive beregninger kan C, C++ og FORTRAN kode lenkes inn og kjøres under kjøretiden av programmet. R kalles ofte et statistikkspråk, men prosjektet selv liker å kalle det et miljø for beregning hvor statistiske metoder er implementert. R er nemlig utvidbart med bruk programpakker. Se også http://www.r-project.org/.

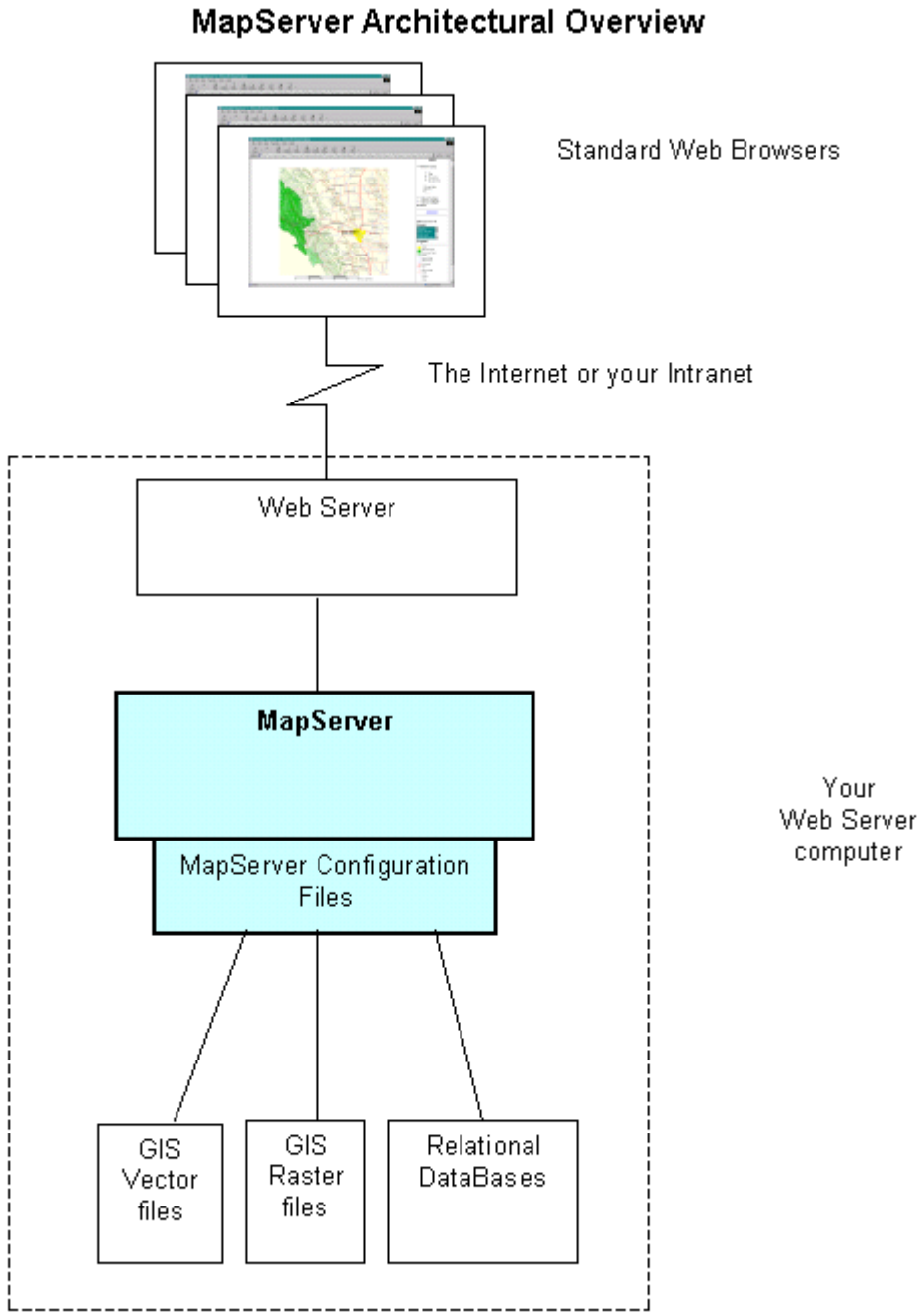
- **Apache HTTP Server** er et samarbeidsprosjekt som har som målsetting å utvikle en vevtjener basert på åpen kildekode. Prosjektet er en del av Apache Software Foundation. Historien om Apache tjeneren startet allerede i 1995 og er tuftet på NCSA httpd versjon 1.3 utviklet av Rob McCool ved ”National Center for Supercomputing Applications, University of Illinois”. Rob McCool sluttet egentlig i sin jobb i 1994, men mange utviklere satt og utviklet programvare for enten å utvide funksjonaliteten eller ”lappe” på eksisterende kode for å tilpasse dette til sitt system. Herrene Brian Behlendorf og Cliff Skolnick satte sammen en e-post liste med den hensikten å få til et samarbeid med de samme utviklerne for å kunne styre videreutviklingen og ha en felles målsetting. På under et år ble ikke bare den første Apache tjeneren utviklet, men den passerte også NCSA httpd 1.3 som nummer 1 på rankinglisten til Netcraft (se http://news.netcraft.com/archives/web_server_survey.html). Der har de ligget siden. Det var først i 1999 at Apache Software Foundation ble stiftet for å gi organisatorisk, juridisk og finansiell støtte til Apache HTTP tjener. Se også http://httpd.apache.org/.

- **GDAL** (Geospatial Data Abstraction Library) er et oversettelsesbibliotek for raster geospatiale dataformater. GDAL er tilgjengelig under lisens for X/MIT(X for X windows er generelt referert til MIT lisens, hvor MIT er akronym for Massachusetts Institute of Technology) og OGC. GDAL biblioteket tilbyr et abstraksjonslag til applikasjoner for håndtering av en rekke romlige rasterformater. Det inkluderer kommandolinjeapplikasjoner for konvertering og håndtering av de samme rasterformatene Se også http://www.gdal.org/.

- **OGR** (OpenGIS Simple Features Reference Implementation) er et C++ bibliotek for "Simple Features" og er et GDAL subbibliotek til som tilbyr et abstraksjonslag for geospatiale data på vektorformat. Det inkluderer kommandolinjeapplikasjoner for håndtering av forskjellige vektorformat som for eksempel ESRI Shapefiles, S-57, SDTS, PostGIS, Oracle Spatial, og Mapinfo mid/mif and TAB format.

- **UMN Mapserver** (UMN er akronym for University of Minnesota) er en karttjener for å lage spatiale Internett applikasjoner. Mapserver er ikke et fullverdig GIS verktøy, og har heller ikke som målsetting å være det. I stedet aspirerer Mapserver til å utmerke seg på datautvelgelse og formatering av spatiale data som kartdata, bilder og vektorgrafikk. Mapserver supporterer skripting i PHP, Python, Perl, Ruby, Java, og C#. Karttjeneren støtter sjablongdrevet utvikling gjennom bruk av sjablongfiler (eng. templates), såkalte MAP-filer. Tjeneren støtter mange raster- og vektorformater. Av andre innbygde egenskaper bør også nevnes elementer for automatisk visualisering av referansekart, karttegnforklaringer (eng. legend) og målestokk. Karttjeneren har også støtte for sanntidsendring av kartprojeksjonen gjennom proj4 biblioteket som igjen støtter tusenvis av projeksjoner. Karttjeneren har innbygget støtte for OGC standardene WMS, WFS, WMC, WCS, Filter koding, SLD, GML, SOS. Se også http://mapserver.gis.umn.edu/.

Figur 10: MapServer arkitektur [http://www.cct.lsu.edu/~gallen/Students/Yadav_2006.pdf]

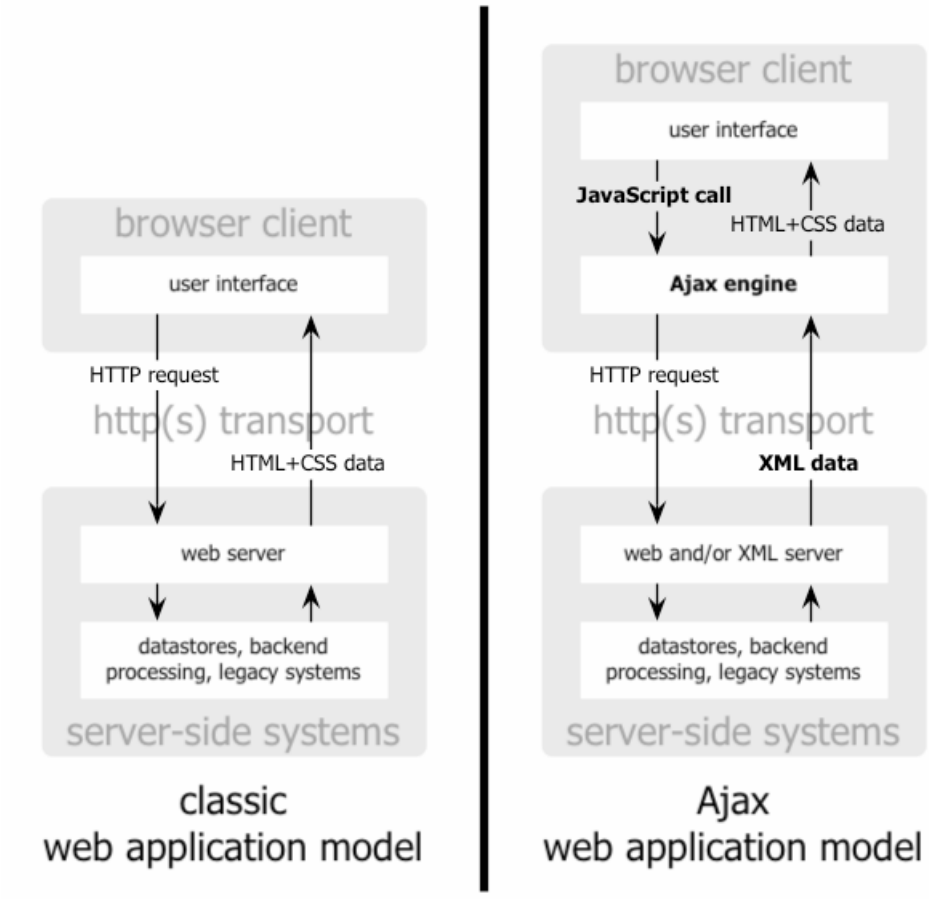


- **Proj4** er et programbibliotek for å konvertere geografisk lengde- og breddegrad (λ,Φ) til kartesiske koordinater (x,y) gjennom en rekke projeksjonsfunksjoner. For en rekke projeksjoner er også den inverse funksjonen implementert. Den originale Proj ble utviklet allerede i 1980 som RATFOR program (en Fortran preprosessor), tuftet på kode fra Geological Survey's General Cartographic Transformation Package (gctp), som siden ble etterfulgt av en mer portabel versjon 2 i 1987. Siden har programbiblioteket blitt kodet om i C (i Kernighan&Ritchie stil) da et annet kartprogram (MAPGEN) ble laget for UNIX. Dette ble gjort fordi Proj var og er en del, av MAPGEN.

- **AJAX** (Asynchronous JavaScript and XML) omfatter teknikker innenfor vevutvikling for å lage interaktive vevapplikasjoner. AJAX er ingen teknologi, men et konsept og en samling av flere teknologier. Den har eksistert i flere år, men det er først i de senere år at vevutviklere har sett nytten i denne teknikken. Det var Microsoft med sitt ActiveX-objekt som først startet med XMLHttpRequest. Versjon 5.x og 6.x av Internet Explorer benytter ActiveX til å lage et XMLHttpRequest-objekt mens Internett Explorer 7.0 vil ha dette som et nativt objekt slik som Mozilla-implementasjonen. Kilde - http://no.wikipedia.org/wiki/Ajax_(Internett).

  AJAX inkluderer følgende elementer:
  - standard presentasjon ved bruk av HTML/XHTML and CSS;
  - dynamisk visualisering og interaksjon ved bruk av DOM;
  - datautveksling og manipulasjon ved bruk av XML and XSLT;
  - asynkron dataoverføring med XMLHttpRequest;
  - og JavaScript som binder alt sammen.

Figur 11: AJAX modellen – en metode for utvikling av dynamiske vevsider med bruk av eksisterende vevteknologi.



Første gang termen ble benyttet offentlig var av av Jesse James Garrett februar 2005.
Se også [http://www.adaptivepath.com/publications/essays/archives/000385.php].

1.6 Komponenter i rammeverket

I mitt rammeverk har jeg valgt GRASS som GIS-verktøy. Dette valget begrunnes med at GRASS er ansett som det mest komplekse GIS-verkøyet blant åpne kildekodeløsninger. Det kan implementeres på flere plattformer, og har en rekke funksjoner som egner seg for håndtering av nedbørdata. GRASS håndterer georefererte data på både raster- og vektorformat i tekstkommandolinjemodus eller via et grafisk brukergrensesnitt. Viktig egenskaper med GRASS er at systemet støtter mange formater og at det er mulig å automatisere samt foreta datautvelgelse med f. eks bruk av skript.

GRASS har et innbygdt databasesystem for lagring av nedbørdata. GRASS SQL er veldig begrenset og er ikke i nærheten av å følge standarden. Det er også andre aspekter i forbindelse med lagring av data i rammeverket, som ikke er mulig eller er tungvint i GRASS. Jeg har derfor implementert PostgreSQL database som en datalagringskomponent i rammeverket. Med utvidelsen PostGIS er PostgreSQL i stand til å håndtere lagring av geospatiale- og temporale data, samt å kunne formatere data på SVG format.
Det finnes andre GIS databaser, men mitt kjennskap til PostgreSQL fra før, samt sitt gode rykte, popularitet og PostGIS utvidelsen gjorde valget enkelt.

Komponenter i GIS løsningen som Apache vevtjener samt tjener/klient prosessering er ”limet” som binder det hele sammen og vil bli diskutert, men i kontekst med de andre hovedkomponentene. Jeg vil drøfte hvordan disse komponentene interagerer med hovedkomponentene for å gi støtte for transport, manipulering, formatering og visualisering i SVG (X3D).
Jeg kan kort nevne at Mapserver er en ypperlig applikasjon å benytte til prosessering på tjenersiden gjennom bruk av CGI (Common Gateway Interface). Programtillegg installert på klienten er strategien for å tolke formaterte nedbørdata dokumenter på XML format.
Jeg vil ikke være bastant i valg av verktøy for prosessering på tjenersiden fordi min hensikt er å støtte all teknologi som kan jobbe sammen med de valgte komponentene, og som vil bidra til å øke funksjonaliteten til rammeverket. Jeg har nevnt Mapserver som et versatilt CGI-verktøy.

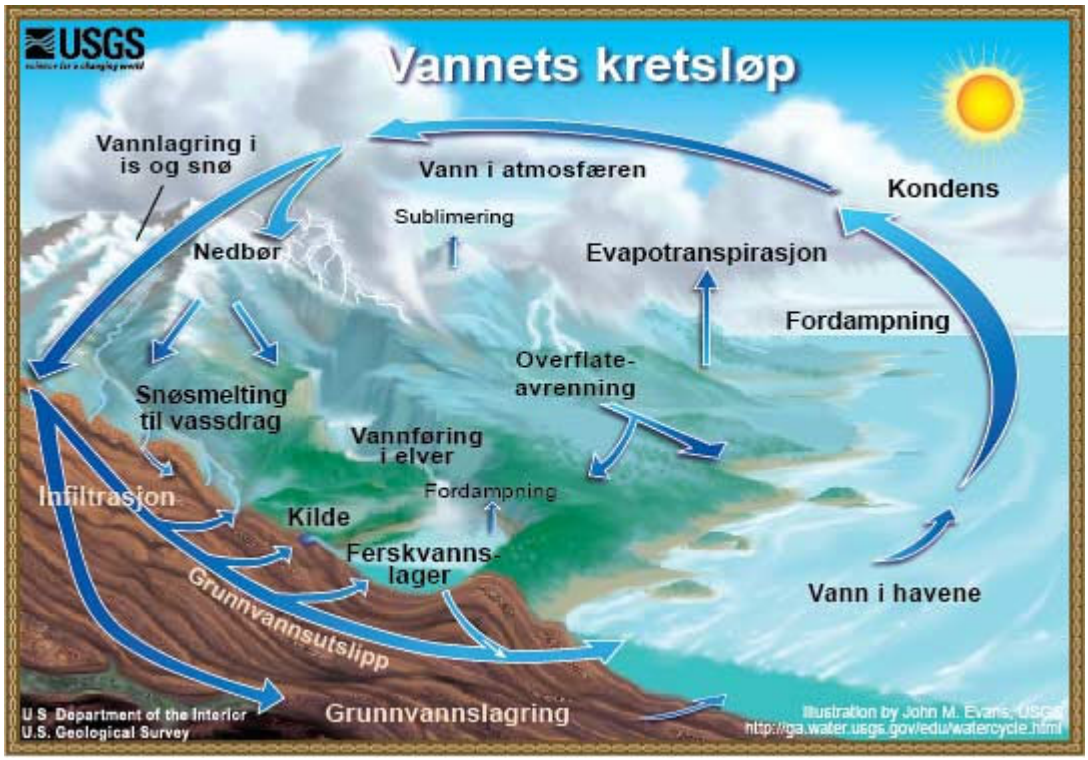
Jeg har valgt å benytte meg av to nettlesere i rammeverket uten bastant å velge den ene eller den andre. Mozilla-Firefox har innebygd støtte for et subsett av SVG standarden og er åpen kildekode. Opera er den nettleseren som har best støtte for SVG. Opera følger en politikk om å være mer ”helhjertet” i sine støtter for åpne standarder – og er således et fornuftig valg.

Hovedkomponentene i rammeverket blir drøftet utover i denne oppgaven, unntatt Apache vevtjener. Apache vevtjener er åpen kildekode, men er ikke det eneste mulige valget av vevtjener. Den er dog populær og har innebygd en støtte for en rekke applikasjoner og dokumentformater. Jeg har derfor mer eller mindre stilletiende valgt å benytte denne tjeneren.

2 Datagrunnlag

For å kunne si noe meningsfylt om atmosfærens framtidige tilstand, "været", må man ha mest mulig informasjon om nåtilstanden. Diagnosen må stilles før man kan komme med prognosen. For å få kunnskap om atmosfærens tilstand, må en foreta observasjoner og målinger. Til dette benyttes forskjellige instrumenter sammen med menneskelig skjønn.
I dette kapittelet skal vi se mer detaljert på datagrunnlaget, nemlig nedbørdata. Nedbør beskriver en tilstand i vannets kretsløp.

Figur 12: Vannets kretsløp. Se http://ga.water.usgs.gov/edu/watercyclenorwegian.html.



2.1 Stasjonsnettet

Observasjoner foretatt ved jordoverflaten, i luften og i havet er grunnlaget for overvåkingen og utgangspunkt for beregningene, sammen med kunnskap om atmosfæresystemet formulert ved prognostiske likninger.
Målinger og bilder fra satellitter er også en viktig del av datagrunnlaget. Disse inkluderer nå også vind og bølger på verdenshavene. Værballonger gir observasjoner fra høyere luftlag og er nødvendige for en tilstrekkelig kartlegging av forholdene i atmosfæren.

Ved årsskiftet 2004/2005 var det i drift 718 meteorologiske observasjonsstasjoner i regi av Meteorologisk institutt. Flere av stasjonene drives i samarbeid med andre virksomheter.

Av disse er ca 200 værstasjoner og 453 nedbørstasjoner. Nedbørstasjoner måler kun nedbør, mens værstasjoner måler flere parametere inkl. nedbør, så det totale antall stasjoner med nedbørmålinger estimeres til ca. 600.

Det nasjonale stasjonsnettet for nedbør kan ikke gi pålitelige data for virkelig nedbør i fjellet. Dels er det vanskelig å drive observasjoner regelmessig uten observatører, og det er derfor svært få nedbørstasjoner over 1000 moh. Dels er oppfangningssvikten for standard nedbørmålere et alvorlig problem overalt, og særlig i fjellet med sterke vinder og mye snønedbør. Vår kunnskap om virkelig nedbør i fjellet bygger derfor i stor grad på indirekte data, fra avløpmålinger i vassdragene og fra snømålinger på breene og andre fjellstrøk. Så stor kan oppfangingssvikten være, at avrenningen i elvene enkelte steder tilsynelatende er større enn nedbøren som er målt i nedbørmålere. Da dette forholdet ble kjent i sin tid, ble det omtalt som "det hydrologiske paradokset" [Tollan, 1993].

2.2 Hva er nedbør?

Skyer består av partikler (vanndråper og/eller iskrystaller) som er så små at oppdrift og tyngdekraft nesten oppveier hverandre. Dermed holder de seg svevende i atmosfæren. Sky-dråpene har en diameter på ca. 0,01 mm.
Forskjellige prosesser inne i skyene kan få noen av partiklene til å vokse (koalesens). Blir de store nok faller de ut av skyene og når bakken. Vi kaller dette for nedbør. Nedbøren registreres i en nedbørmåler.
Nedbørformen avhenger av temperaturen i lufta nær bakken, der nedbørmåleren er plassert og av de fysiske prosessene som foregår inne i skyene. Se
http://met.no/met/met_lex/l_p/nedbor/index.html.

2.3 Nedbørmålinger

Hva mener vi når vi sier at i løpet av siste døgn har det falt så og så mange millimeter nedbør? Nedbørmengde måles i antall millimeter. Begrepet nedbørhøyde er et synonymt begrep.

Meteorologisk terminologi:

Nedbørhøyde er den høyde som nedbøren (omgjort til vann) ville stige til om den falt på flatt underlag og ikke rant bort. Er nedbørhøyden 10 millimeter så menes det at vannet ville stå 10 millimeter overalt på bakken. På en flate på 1 x 1 m vil det altså være falt 100 x 100 x 1 = 10 000 cm^3 = 10 liter vann. 1 cm nysnø svarer omtrent til 1 mm regn.

Arealet som nedbøren faller ned over, kalles nedslagsfeltet i overført betydning fra faget hydrologi. Areal er da et av attributtene til et slikt område.
Når vi skal omgjøre nedbørmengden til antall mm, så er regner man ut fra de facto meteorologisk nedslagsfelt for nedbør 1 x 1 m stort.

I Norge måles nedbør med bruk av manuelle- og automatiske nedbørmålere. Felles for begge er at målerne består av en sylindrisk beholder en åpning i toppen med kjent horisontalt

tverrsnitt målt i cm^2. Tverrsnittet utgjør et nedskalert et nedslagsfelt i forhold til definisjonen. Vanlig tverrsnitt eller åpning på målere som benyttes i Norge er hhv. 200 og 220 cm^2. Oppløsningen på målingene er 1/10 millimeter.
Nedbørmåleren monteres på et stativ eller pidestall slik at åpningen befinner seg 1.5 – 2 meter over bakkenivå. Denne høyden er en internasjonal standard som er vedtatt, ut fra en viten om at vinden påvirker oppfangsevnen og at vindstyrken øker med høyden. Vindstyrke "nær" bakkenivå kan veldig forenklet beskrives ut fra en logaritmisk vindprofil eller vindgradient. Så er det sikkert noen som allerede tenker at der er målt snødybder på over to meter i dette landet. Ja, det er det, og på slike utsatte plasser må man inngå kompromisser i forhold til standarden og heve pidestallen over antatt maksimal snødybde. Denne høyden må eksplisitt gjøres kjent til alle som benytter data fra måleren. Jeg har allerede nevnt vind som en påvirkningsfaktor, og det er derfor vanlig at det monteres vindskjerm rundt beholderen for å redusere påvirkningen fra vinden.
Eksponering til en nedbørmåler er viktig. Den bør plasseres i mest mulig åpent lende vekk fra obstruksjoner som f. eks bygninger og trær.

Ved bruk av manuelle nedbørmålere tappes innholdet opp i et skalert måleglass.
Volumet i måleglass og beholder det samme, og forholdet mellom arealet eller diameteren av beholderen og glasset benyttes til å beregne nedbørmengden.
$V = A * h = Am * hm$, hvor Am er arealet av måleglasset i cm^2 og hm er høyden av nedbøren i måleglasset i cm. Enheten for V er her cm^3 (≈ml).
Dersom nedbørsmengden er målt som høyden i et (rett) glass blir nedbørmengden i mm nedbør: $h\ (mm) = (Am / A) \times h_m / 10$.
Ved nedbør i form av snø tines snøen før tapping i måleglass. Under tining må man passe på å ikke tilføre for mye varme, slik at ikke nedbøren fordamper og målingen blir feil.

En type automatiske nedbørmålere baserer seg på vektmåling. Automatisk er et relativt ord i denne sammenhengen. Det som menes er at målingene leses elektronisk med en vektsensor og en signalomformer til en datalogger. Selve måleren er slik utstyrt at den krever minimalt med ettersyn, men innholdet i beholderen må tømmes manuelt når den begynner å bli full.
I vintersesongen fylles beholderen med en frostvæskeblanding som har samme tetthet som vann. Det er for å hindre at vannet skiller seg fra frostvæsken og fryser til is. For å hindre fordamping fyller man i tillegg på noen desiliter av en bestemt olje med høy viskositet som legger seg på overflaten av innholdet i beholderen. Mengden er nøye tilpasset for å sørge for at nedbøren faller igjennom oljelaget og blander seg med frostvæsken og forblir eller blir til flytende form avhengig av om nedbøren er i form av regn, snø eller hagl.

Formelen for tetthet er $\rho = m / V$, hvor m er masse og V er volum. Tettheten til vann er ca. 1 g/cm^3. A er kjent og m måles av vektsensoren i enheten cm^3 (≈ml).
Sammenhengen mellom masse og nedbørmengden i mm nedbør: $h\ (mm) = m / A = (10 \times m)/A$.

En annen type automatiske nedbørmålere er såkalte vippepluviografer, som fra åpningen og nedover danner en trakt. Under traktmunningen er det plassert en vippe med to "begre" på hver side. Disse er mest egnet til å måle nedbør i flytende form (regn, yr) så sant de ikke har innlagt et varmeelement (feilkilde ved fordamping av nedbør som blir liggende i trakten). Nedbøren faller ut av traktmunningen og oppi et av begrene som har en kjent volumkapasitet. Når begeret er fullt vippes det ned og innholdet tømmes ut, mens det andre starter å fylles. Prinsippet med målingene er å telle antall vipper N gjennom en tidsperiode og nedbørhøyden

kan da beregnes som h(mm) = N x (10 x V/ A). V er da volumet til begrene enheten cm^3 (≈ml). Begrenes volumkapasitet er da også et mål for oppløsningen på målingene.

MI har i de siste årene montert (delvis i samarbeid med andre institusjoner) flere værradarer langs kysten av Norge. En værradar er en Doppler-radar som i tillegg til å observere ekko fra nedbør også måler nedbørområdenes hastighet. Dette gjør det mulig å også si noe om vindretningen og vindstyrken.
En værradar må altså ha best mulig sikt, men bør likevel ikke ligge særlig høyere enn 500–600 meter over havet. På grunn av jordkrumningen øker radarstrålens høyde over det nivået værradaren er plassert på med økende avstand. Dersom en værradar blir plassert for høyt og har for stort overvåkningsområde, er det derfor risiko for at nedbør kan ligge under radarstrålen og dermed være «usynlig» eller ligge så høyt at det er vanskelig å si noe om hvilke konsekvenser den vil gi på bakkenivå. For å sikre datakvaliteten er derfor hver værradar i Norge gitt et overvåkningsområde begrenset til 240 km. Se http://met.no/met/met_lex/v_a/varradar/index.html.
Foreløpige resultater tyder på at kvantitativ estimering av nedbørsmengde er beheftet med en rekke usikkerheter.

2.4 Rapportering av nedbørobservasjoner

UTC er et akronym for Coordinated Universal Time og angir grunnlaget for sivile tidsangivelser i alle land. UTC erstatter det tidligere GMT (Greenwich Mean Time) og ligger 1 time før norsk normaltid (2 timer før norsk sommertid). På norsk skriver vi gjerne universell tid UTC dersom vi ønsker å presisere hva som menes med de tre bokstavene UTC. Som tidsangivelse angir UTC døgnets timer fra 00 til 23. I værvarsling er bruk av UTC nødvendig bl.a. for å kunne arbeide med data fra land i ulike tidssoner.

Et nedbørdøgn er tiden fra kl 06 UTC til kl 06 UTC neste dag. Dette tilsvarer fra kl 07 til 07 neste dag norsk normaltid og fra kl 08 til 08 neste dag norsk sommertid. Alle norske, meteorologiske stasjoner måler nedbør på dette tidspunktet. Noen måler i tillegg kl 00, 12 og 18 UTC.

Manuelle nedbørmålinger utføres av en observatør som enten sender inn et utfylt registreringsskjema eller bruker mobiltjenesten SMS (Short Message Service) for mer sanntidsrapportering. En observatør forholder seg til lokal tidsangivelse som definerer nedbørdøgnet fra kl 08 til 08 neste dag ved sommertid og fra kl 07 til 07 neste dag ved normaltid.

Ved automatiske målinger sendes data enten via GSM (Global System for Mobile Communications) nettet eller samles inn via oppringt samband til MI. Et modem (GSM eller analogt) besørger kommunikasjonen med datalogger. Data overføres med bruk av proprietære protokoller eller som ren tekst. Tekstformatet benyttes når dataloggeren selv initierer en forsendelse av data som SMS.
Automatiske stasjoner sender statistiske data til MI hver time. Med det menes at dataloggeren utfører en del parameterberegninger i stedet for å sende enkeltmålinger.

Data blir enten kodet i synoptisk kode (SYNOP surface synoptic observations) eller rapporteres i som tall i med enheter definert i SI systemet.
Observasjonene dekodes og legges inn i klimadatabasen ved MI. De fleste opplysningene legges inn i klartekst (f. eks nedbør), men for noen parametere brukes tallkoder.

2.5 Klimadata

Klimadivisjonen ved MI er ansvarlig for produksjonen av klimadata. Klimaet er værforholdene over lengre tid beskrevet ved hjelp av statistiske verdier. Grunnlaget er observasjoner av været over en så lang periode at enkeltvise ekstreme værsituasjoner ikke påvirker resultatene vesentlig. En kan også si at klimaet er et konsentrat av været.

Hva er normalt?

De vanligste klimaverdiene som lages ut fra observasjonene er middelverdier og variasjoner rundt disse. Middelverdier eller gjennomsnittsverdier for bestemte 30 års perioder som 1901–1930, 1931–1960 og 1961–1990 kalles normaler. Det er en internasjonal avtale om at normalene skal benyttes som offisielle middelverdier slik at det blir likt over hele verden. Nå benyttes normalene for 1961–1990. Se http://met.no/met/met_lex/l_p/index.html#N.

Interpolering av nedbørdata avleder to temporale typer statistiske verdier, nemlig døgn- og månedsnedbør.
Klimadata som utgjør basisdata for kartproduksjon, lagres i dag i en Oracle database for hver stasjon. Klimadata lagres som tidsserier basert på døgnstatistikk og månedsstatistikk. Dette er hensiktsmessig format sett ut i fra intern produksjon og rutiner hvor disse dataene blir benyttet. Klimadata kan gis for andre tidsperioder enn døgn- og månedsstatistikk, men de er da avledet av disse igjen.

For produksjon av klimakart basert på nedbørdata benyttes to metoder for interpolering.

Kartdata basert på døgnnedbør er utviklet med bruk av TIN (Triangulated Irregular Network) modellering. TIN metoden som er valgt kalles DT (Delaunay triangulering), og kan beskrives som utvikling av trekantflater med høydekorreksjon.
Jeg har ikke benyttet meg av døgnnedbør i min oppgave, men vil bare nevne hvordan dette er gjort fordi det er relevant metode for geometrisk modellering i GIS. Se geometrisk modellering i kapittel 5.

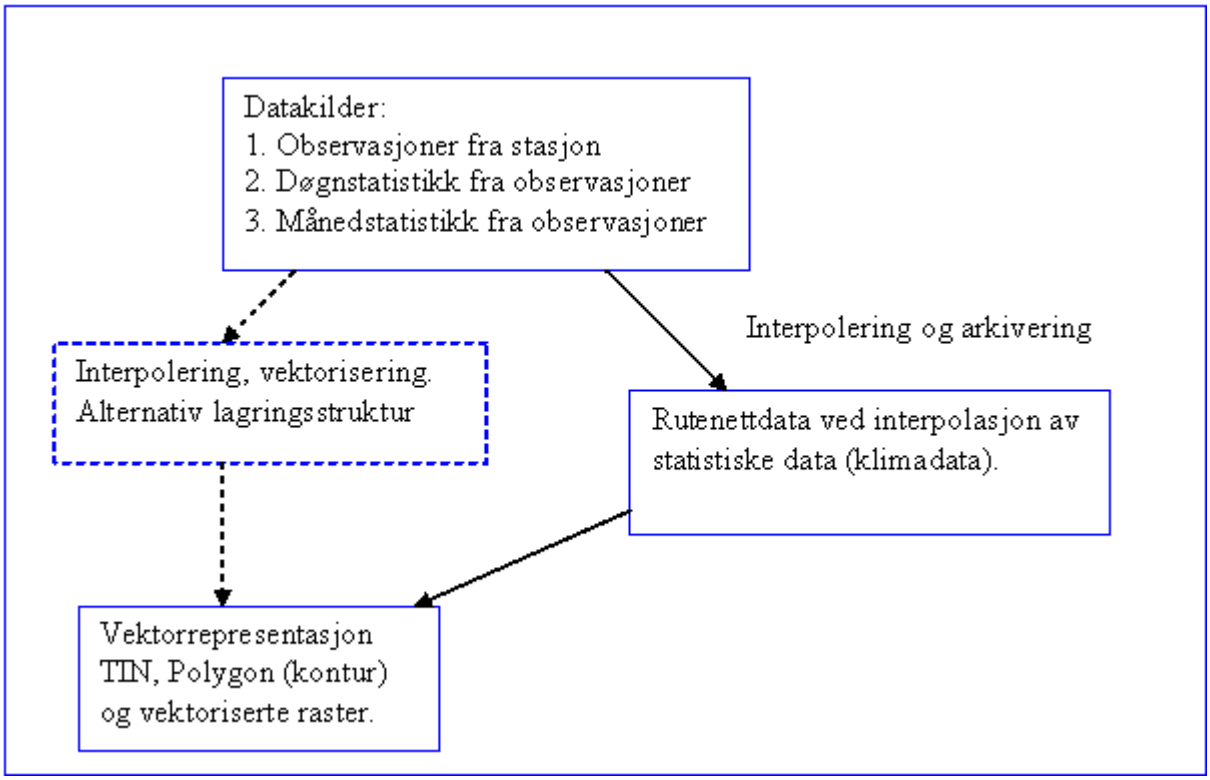
Kartdata basert på månedsnedbør er avledet med en metode som kalles ANUDEM (se http://cres.anu.edu.au/outputs/anudem.php), og er utviklet ved Centre for Resource and Environmental Studies The Australian National University.
Metoden beregner og avleder et regulært grid DEM (Digital Elevation Model). Inndata er for eksempel høydepunkter, høydekonturer og polygoner. Metoden er en egen spline variant basert på avvik i forhold månedsnormalen i prosent. Denne metoden er spesielt utviklet for hydrologiske formål.
Månedsnedbør lagres på ESRI binær grid format. Rutenettet består av ruter med geografisk utstrekning på 1 x 1 km (omtrentlig 30 x 30 buesekunder i geodesiske koordinater), som er vurdert som optimalt for de fleste forhold som ikke er avhenging av data fra spesifikke stasjoner. Interpolasjonsmetodikken baserer seg på terrengmodeller og geografisk avstand til

observasjonsstasjonene. Endringer i stasjonsnettet som følge av nedleggelse eller etablering av stasjoner influerer ikke på interpolasjonsmetodikken, så sant spredningen av stasjonene ikke endres i vesentlig grad.

Begge metoder er implementert på MI ved hjelp av et proprietært høynivåspråk kalt AML (ARC Macro Language). AML er utviklet av ESRI for å lage sluttbrukerapplikasjoner for ArcInfo arbeidsstasjoner.

Figur 13 illustrerer en mulig og en eksisterende vei fra lagrede klimadata til visualisering hos instituttansatte og offentligheten [3].

Figur 13: Skjematisk tegning av produksjonslinjen



I mitt arbeid har jeg bestemt meg for å benytte ett år med interpolerte nedbørdata for 2005. Det er nedbørdata utlagt som et rutenett begrenset til fastlandet. Svalbard og nærområdene samt området rundt Jan Mayen er altså ikke med i datasettet.

2.6 ESRI binære rasterformat

ESRI 2D binære grid format er egentlig en hierarkisk filmappestruktur, med underordnede filer i mappen for å beskrive alle data i rutenettet.

3 Kommentarer til Figur 13
Heltrukne linjer angir produksjon av klimadata og datagrunnlaget slik det foreligger i dag. Stiplet linje angir produksjon av datagrunnlaget hvis man ønsket å integrere gridding inn i rammeverket.

Informasjon på engelsk beskriver hva filene er kalt og deres respektive funksjonalitet og innhold. Se også http://home.gdal.org/projects/aigrid/aigrid_format.html.

- **dblbnd.adf**: Contains the bounds (LLX, LLY, URX, URY) of the portion of utilized portion of the grid.
- **hdr.adf**: This is the header, and contains information on the tile sizes, and number of tiles in the dataset. It also contains assorted other information I have yet to identify.
- **sta.adf**: This contains raster statistics. In particular, the raster min, max, mean and standard deviation.
- **vat.adf**: This relates to the value attribute table. This is the table corresponding integer raster values with a set of attributes.
- **w001001**.adf: This is the file containing the actual raster data.
- **w001001x**.adf: This is an index file containing pointers to each of the tiles in the w001001.adf raster file.

Ovennevnte standard filer beskriver ESRI binære rutenettformat. I tillegg kan formatet inneholde filen **prj.adf**, som beskriver kartprojeksjonsdata for rutenettet.

Vi kan liste ut mer spesifikk geografisk informasjon om innholdet i rutenettet ved å kjøre applikasjonene som medfølger GDAL biblioteket.
Applikasjonen `gdalinfo` henter ut geografisk informasjon fra rutenettet med nedbørdata. Dette er informasjon man siden trenger for å etablere et arbeidsområde i GRASS for bearbeiding av nedbørdata. I Unix/Linux aktiverer man manualsidene for applikasjonen med ”`man gdalinfo`”.

Det er to måter å liste ut informasjonen på:

- gdalinfo hdr.adf
- gdalinfo jan_2005/

Kommandolinjeopsjon i punkt 2 hinter til tidligere diskusjon om ESRI binær rutenettformat som en overordnet filmappestruktur.

Informasjonsutlisting:

```
Driver: AIG/Arc/Info Binary Grid
Size is 1216, 1551
Coordinate System is:
PROJCS["UTM Zone 33, Northern Hemisphere",
    GEOGCS["WGS 84",
        DATUM["WGS_1984",
            SPHEROID["WGS 84",6378137,298.257223563,
                AUTHORITY["EPSG","7030"]],
            TOWGS84[0,0,0,0,0,0,0],
            AUTHORITY["EPSG","6326"]],
        PRIMEM["Greenwich",0,
            AUTHORITY["EPSG","8901"]],
        UNIT["degree",0.0174532925199433,
            AUTHORITY["EPSG","9108"]],
        AXIS["Lat",NORTH],
        AXIS["Long",EAST],
        AUTHORITY["EPSG","4326"]],
```

```
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",0],
    PARAMETER["central_meridian",15],
    PARAMETER["scale_factor",0.9996],
    PARAMETER["false_easting",500000],
    PARAMETER["false_northing",0],
    UNIT["METERS",1]]
Origin = (-75755.000000,8000500.000000)
Pixel Size = (1000.00000000,-1000.00000000)
Corner Coordinates:
Upper Left  (  -75755.000, 8000500.000) (  1d21'1.64"W, 71d23'59.64"N)
Lower Left  (  -75755.000, 6449500.000) (  5d17'21.41"E, 57d48'51.98"N)
Upper Right ( 1140245.000, 8000500.000) ( 33d 4'24.76"E, 71d14'16.08"N)
Lower Right ( 1140245.000, 6449500.000) ( 25d46'39.17"E, 57d43'38.24"N)
Center      (  532245.000, 7225000.000) ( 15d41'15.26"E, 65d 8'48.77"N)
Band 1 Block=256x4 Type=Float32, ColorInterp=Undefined
  Min=2.944 Max=1007.231
  NoData Value=-3.40282e+38
```

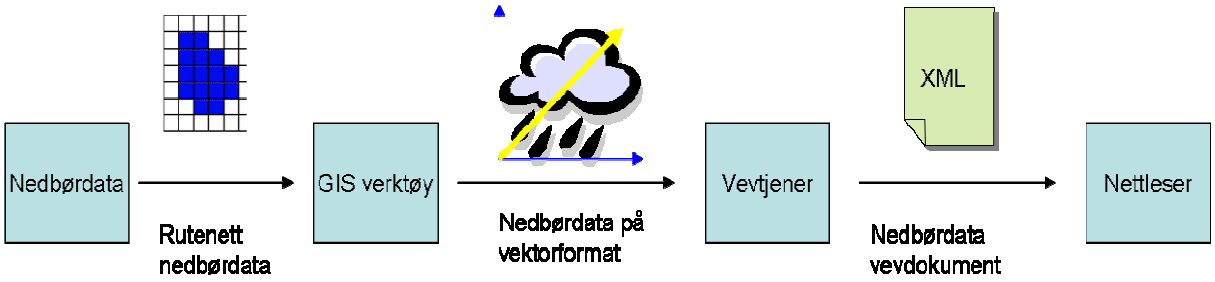
Fra listingen ser vi at kartprojeksjon er UTM-sone 33. Bemerk at Norge består av flere UTM-soner. Det pragmatiske valget er å finne UTM-sonen som beskriver det geografiske tyngdepunktet av kartflaten for Norge. Se neste kapittel om kartprojeksjoner. Unøyaktigheten som følge av overlappingen av UTM-soner medfører, er akseptabel for produksjon av klimakartobjekter. I mitt arbeid har jeg adoptert tilnærmingen som er gjort av Klimadivisjonen.
Størrelsen på rutene er (Pixel Size = (1000.00000000,-1000.00000000)) som altså er 1000 x 1000 angitt med SI enheten meter. Se kommandolinjeopsjon UNIT i manualsidene.

Datumet og ellipsoiden er WGS84 (se kapittel 4). Listingen angir mininimum- og maksimumsverdi på interpolerte nedbørhøyder i Norge januar 2005, som hhv er 2.944 mm. og 1007.231 mm. At dette er interpolerte verdier, vil man forstå når oppløsningen på selve målingene er utført med 1 desimals nøyaktighet.
EPSG (European Petroleum Survey Group) kodetall er en standard for å beskrive forskjellige koordinatsystemer. Standarden inneholder også transformasjonsinformasjon for å konvertere etablerte koordinatsystemer. Mer om EPSG følger i kapittel 4 hvor jeg gir en basal innføring i kart og kartmodeller.
Av annen informasjon i listingen kan nevnes parametere, som beskriver jordmodellens flattrykning (298,257223563), store halvakses dimensjon (6 378 137 meter).
Listingen inneholder og informasjon om NULL data dvs. punktdata som ikke representerer noen form for nedbørinformasjon.
Størrelsen på hele rutenettet er 1216 x 1551 punkter, hvilket vil si en geografisk utstrekning på 1 886 016 km^2.
Hjørnepunktene for hele rutenettet er oppgitt i to koordinatsystemer, nemlig UTM og geodesisklengde- og breddegrad-koordinater, hvilket også tilsvarer maksimumsverdiene for himmelretningene nordvest, nordøst, sørøst, sørvest. Vi leser tilsvarende koordinatinformasjon for sentroiden i rutenettet.

3 Systemarkitektur

I dette kapittelet skal vi se nærmere på systemarkitekturen til rammeverket for visualisering av nedbørdata på XML formatene SVG, X3D samt WFS. Rammeverket kan beskrives som et vevbasert GIS arkitektur.

Figur 14: Overordnet arkitektur



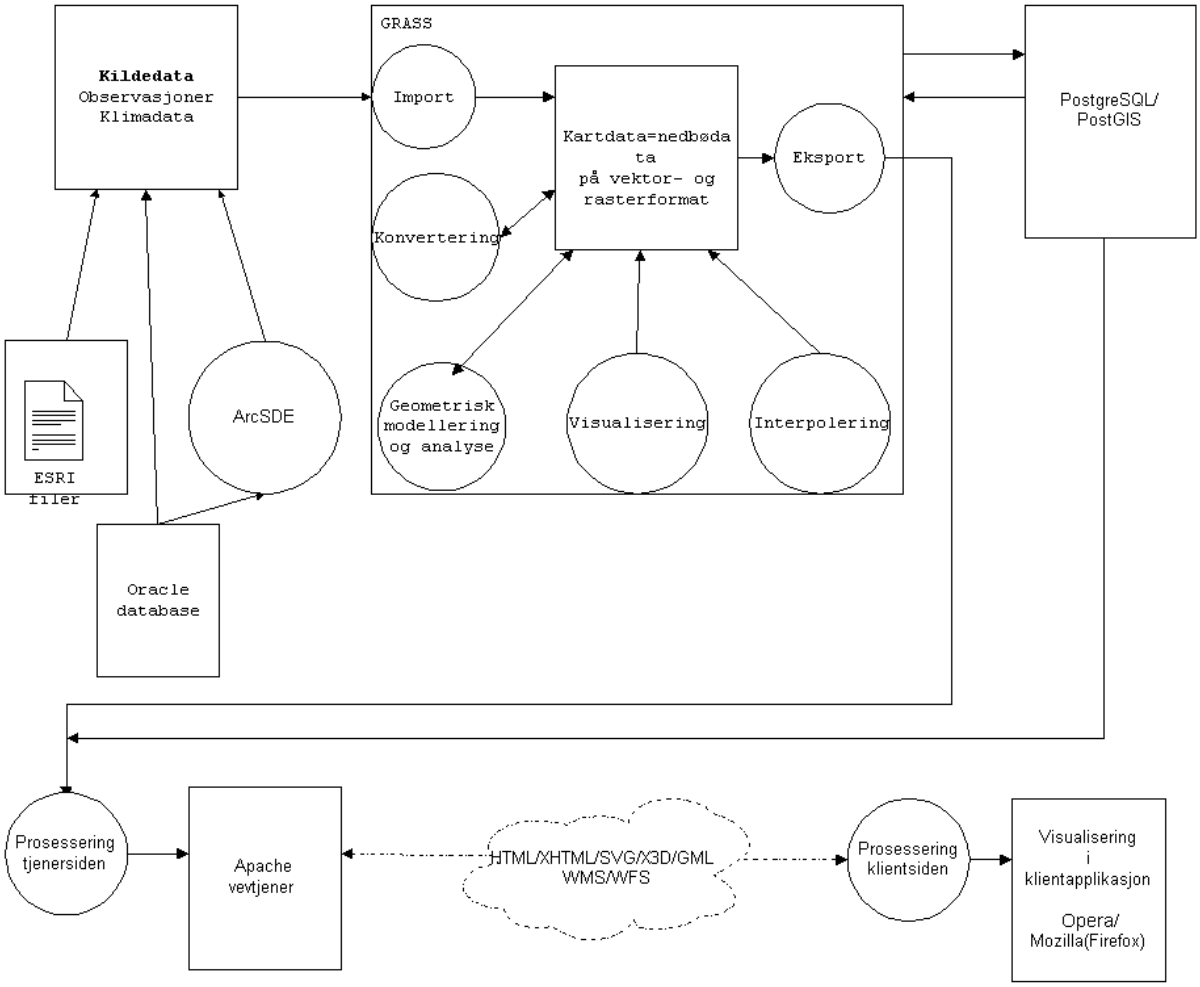
Systemet kan importere, transformere og analysere kildedata i dette tilfellet nedbørdata. Nedbørdata kan lagres på både permanent og/eller temporær basis. Systemet gir mulighet for datautvelgelse i et lokalt intranett og Internett fra en nettleserklient. Dette er en grov beskrivelse av arkitekturen. I de påfølgende avsnitt skal vi se mer detaljert på valgte systemkomponenter.

Systemet består av de valgte hovedkomponentene GRASS, PostgreSQL/PostGIS og Apache vevtjener.

Interpolerte nedbørdata har vi drøftet tidligere og sees her på som inndata. I min realisering av denne arkitekturen har jeg valgt GRASS og PostgreSQL som GIS verktøy og Apache som vevtjener. Klientapplikasjoner som jeg har benyttet for uttesting av systemet er Mozilla Firefox og Opera nettleser. Figur 14 skjuler tjener– og klientprosesser som er ”limet” som binder det hele sammen.

Figur 15 viser arkitekturen i detaljer med de valgte komponenter.

Figur 15: vevbasert GIS arkitektur.



3.1 Ord og forklaringer

I drøftelsene fremover vil jeg bruke kartdata synonymt med nedbørdata for å fremheve at jeg skal lage interaktive kartdokument for å oppfylle en av målene for oppgaven. Drøftelser av aktuell problematikk vedrørende visualisering forringes ikke av den grunn. Visualisering av kartdokument byr på mange og allsidige utfordringer. Jeg vil derfor kort introdusere ord og begreper som brukes videre i oppgaven for å etablere en felles forståelse for bruken av disse begrepene.
Her følger et relevant utvalg av GIS termer hentet rammeverksdokument for prosjektet Norge digitalt, som er en arbeidsgruppe for rammeverk og infrastruktur for stedfestet informasjon i Norge [(Teknologiforum), 2006].

Termene er forsøkt skrevet i henhold til ISO 7004_2000 Retningslinjer for definisjoner. Flere av disse er basert på engelske definisjoner, men er her gjengitt på norsk. Referanse til hvor definisjonene kommer fra er satt i hakeparentes.

- **Geodata** er data om objekter (vann, hus, veger, fyr osv), hendelser og forhold der posisjonen (sted på jorda) er en vesentlig del av informasjonen. [basert på St meld nr 30 Norge digitalt et felles fundament for verdiskaping]
- **Geoportal** enhver portal som vil ha nytte av geografisk informasjon, enten i seg selv, som meny eller for å relatere annen informasjon til geografisk sted eller til andre data.
- **Interoperabilitet** evne til å kommunisere, kjøre programmer, eller overføre data mellom ulike funksjonelle enheter på en slik måte at brukeren ikke trenger spesiell kunnskap om disse enhetenes karakteristikk. [ISO 19118]
- **Kartinnsyn** et eller flere kartlag vist sammen som en ordnet lagpakke**.** Kartinnsynet er gjerne et brukertilpasset kart basert på en eller flere forespørsler, og hvor kartlagene er avpasset kartografisk til hverandre.
- **Kartdata** geodata tilrettelagt for kartproduksjon [Kvalitetssikring av oppmåling, kartlegging og geodata]
- **Kartlag** en eller flere karttema som frembringes i én forespørsel mot en tjenermaskin. Formatet på kartlaget er avhengig av standarden som benyttes.
- **Kartbilde** kan brukes i forbindelse med et resultat fra en ”GetMap-forespørsel” i WMS – standarden eller et resultat basert på flere forespørsler som tjeneren har slått sammen føre leveranse til klienten. Lovlige formater for kartbilder beskrives i standarden [ISO 19128]
- **Kartobjekt** en visuell (kartografisk) presentasjon av synlige eller ikke-synlige geografiske fenomener, f. eks henholdsvis bygning og kommune.
- **Karttema** kartobjekter av samme objekttype og klassifikasjon Merknad: Karttema tilsvarer 'Layer'-begrepet i WMS-standarden. Eksempel: Punktobjekter klassifisert som bygning.
- **Karttjeneste** sammensatt kartinformasjon tilpasset et formål. Graden av funksjonalitet kan være fra ferdigtilpassede visninger av kartinnsyn og egenskapsdata til brukerstyrt visning og redigering av data. Merknad: Karttjeneste kan brukes på klient- eller tjenernivå:
    - Karttjeneste (tjener): En tjeneste på en tjenermaskin som leverer kartlag til klienter, for eksempel en WMS-tjeneste.
    - Karttjeneste (klient) Klientapplikasjon med funksjonalitet og undertjenester (for eksempel WMS-tjenester, søketjenester) som frembringer sammensatt informasjon tilpasset et formål.

3.2 Kildedata

Kilden til kartdata er geodata på rasterformat (nedbørdata i dette tilfellet) fra et kommersielt GIS produkt som drøftet i forrige kapittel. Det er et funksjonelt krav fra oppdragsgiver at systemet må støtte import av nedbørdata fra dette systemet. Kartdata på forskjellige rasterformat, vektorformat og tekstformat kan importeres inn i systemet. Systemet er dermed ikke bundet til en kilde eller til et bestemt produkt.

3.3 GRASS

GRASS (Geographic Resources Analysis Support System) utfører en rekke oppgaver. Disse oppgavene er:

- import av nedbørdata
- lagring både for grafiske data og egenskapsdata
- editering, transformering og oppdatering
- søk basert på elementenes lokalisering og egenskaper inkludert temporale data
- analyse
- presentasjon av analyseresultater
- eksport av nedbørdata
- automatisert produksjonen av karttema basert på nedbørdata

GRASS er ansvarlig for majoriteten av karttemaproduksjonen.

Oppgavene forklart i samme rekkefølge som kulepunktene over:

Import av nedbørdata (på rasterformat) er mulig i GRASS med applikasjoner som benytter det åpne kildekodebiblioteket GDAL (kort beskrevet i kapittel 1). Data arkiveres internt på GRASS rasterformat.

GRASS støtter lagring av både rasterformat og vektorformat, hvor geografisk informasjon separeres fra egenskapsdata. Internt databasesystem for lagring egenskapsdata er basert på dBase filformat realisert gjennom en intern driver kalt DBF. Ulempen med den interne databasen er at den har en meget begrenset støtte for SQL. Ulempen kan kompenseres ved at GRASS har drivere for andre DBMS (database management system) deriblant PostgreSQL. Samarbeidet med ekstern DBMS kan realiseres på to måter:

- lagring av egenskapsdata i ekstern DBMS og geografisk informasjon i GRASS
- lagring av både egenskapsdata og geografisk informasjon, dvs. full eksport av data

Dette gjør det mulig å hente ut data med hjelp av SQL.
Uheldigvis har ikke GRASS innebygd støtte for temporale data på vektorformat. Rasterdata kan derimot gis et tidsstempel. I skrivende stund er behovet temporale data for vektordata aktualisert av brukere av GRASS, og GRASS utviklere arbeider med å implementere et løsningsforslag i fremtidige versjoner.

Editering og transformering er mulig for både rasterdata og vektordata.
Interaktiv editering og transformering er typisk en øvelse man gjør for å utvikle kartlagene man ønsker. Relevante oppgaver er for eksempel å definere farger som skal assosieres med egenskapsdata, verdiformatomforming, for eksempel å konvertere data fra flytall til heltall, noe som er relevant å utføre med nedbørdata.
GRASS har innebygde og automatiske metoder for konvertering av raster til vektorformat. Vektorisering av nedbørdata gjøres i GRASS.

Søk basert på temporal informasjon er ikke uten videre mulig i GRASS som nevnt i avsnittet om lagring. Ved å lage egne løsninger og f. eks. implementere et temporalt attributt i

PostgreSQL er det likevel mulig å realisere temporale søk. Søk basert på egenskapsdata og lokasjon er en triviell affære med bruk av SQL og andre databaseapplikasjoner i GRASS og mot andre DBMS.

Det er en rekke metoder for å foreta analyse av rasterdata og vektordata. Analyse er et argument for å lagre rasterdata permanent som kilde til karttemaproduksjon. Enkelte analysealgoritmer er med hensikt basert på rasterdata, fordi det forenkler algoritmen. Andre analysealgoritmer kan forenkles når nedbørdata er på vektorformat. Det er også slik det er tenkt og implementert i GRASS.
Relevante analysemetoder er kartalgebra, klassifisering, resampling, uhenting av celleverdier fra rasterdata og rasterisering av vektorformat (omforme vektordata tilbake til rasterformat). Konturering er f. eks en metode som kun utføres på rasterdata. Andre analysemetoder av nedbørdata på vektorformat inkluderer Delaunay triangulering og topografisk analyse (isolinjer, stigning på flater, retning på flater).

Visualisering av karttema er mulig i GRASS. GRASS har innebygde grafiske drivere som støtter de grafiske vindussystemene X Windows og Microsoft Windows. Karttema på både vektorformat og rasterformat kan visualiseres. I tillegg kan karttema eksporteres til PNG (Portable Network Graphics) rasterfilformat med hjelp av en PNG driver, for bruk i andre applikasjoner. I mitt arbeid har jeg benyttet visualisering interaktivt for verifisering av kartbilder etter at editering, transformasjon og analyse er utført.

Eksport av nedbørdata, både på raster- og vektorformat, er mulig i GRASS. I mitt arbeid har eksport av vektordata vært relevant med tanke på viderebehandling, lagring og visualisering hos klienten.
Eksport av vektordata til PostgreSQL har jeg vurdert og testet som et alternativ til å inkludere temporal informasjon til kartdata. GRASS-metodene for eksport av data på vektorformat er realisert gjennom OGR åpen kildekodebibliotek, som er nevnt i kapittel 1.

Automatisering av alle ovennevnte oppgaver er fullt ut mulig. Dette er helt essensielt for at systemet skal være dynamisk.
En stor ulempe med GRASS er at produktet kun støtter sekvensiell klienttilgang til et kartdatasett. En bruker kan kjøre en instans av GRASS kartdatasett ad gangen. Samme bruker kan derimot samtidig kjøre en GRASS instans mot et annet kartdatasett. Det er dog en mager trøst, da det logiske og mest hensiktsmessige er å samle kartdata med samme tematiske og temporale egenskaper i ett kartdatasett.
Manglende muligheter for å håndtere samtidige tilkoblinger kompliserer løsninger for automatisering i GRASS. En løsning som ikke håndterer dette vil resultere i en avbrutt operasjon og feilmelding. Å lage en løsning som setter klienten ”på vent” øker kompleksiteten i programkode og bidrar bare til at responstiden øker med antall brukere. Det er en problemstilling man på en eller annen måte må håndtere. I en vevapplikasjon vil det typisk være en ”bruker” (vevbrukeren eller en predefinert standardbruker) som gjør en rekke spørringer. Det er en mulig måte å håndtere dette på. Se videre drøftelser som berører problemstillingen i neste punkt.
Jeg henviser til kapittel 6 for en detaljert beskrivelse og bruk av GRASS.

3.4 PostgreSQL/PostGIS

Jeg vil her kort drøfte aktuell og alternativ bruk av PostgreSQL som kan håndtere romlige data og spørringer med utvidelsen PostGIS. Jeg henviser også til kapittel 7 som gir en introduksjon til PostGIS og PostgreSQL.

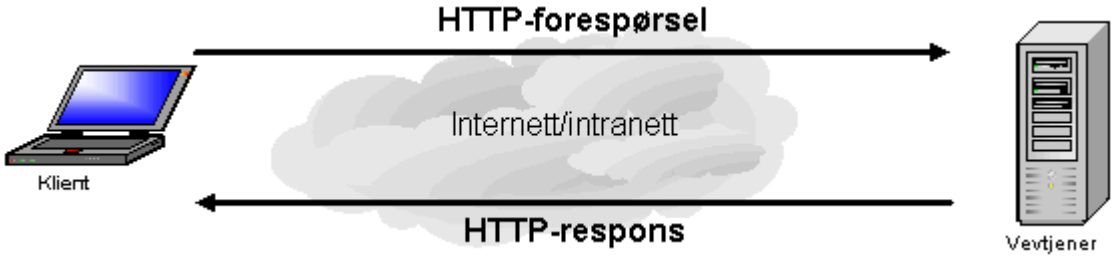
Jeg har allerede nevnt muligheten for å bake inn et temporalt attributt. En viktig fordel er bruk av det standardiserte spørrespråket SQL for romlige spørringer. SQL er som jeg allerede har nevnt meget begrenset implementert i GRASS sitt interne databasesystem. Fordelen med bruk av SQL er at dette støttes av mange språk via egne databasedrivere. Eksempelvis kan vi nevne språk som Java, Perl, C/C++, Python, PHP samt en rekke åpne kildekode applikasjoner.
En åpenbar fordel er at administrasjon av data og brukere er enklere i PostgreSQL enn i GRASS. Et viktig aspekt med PostgreSQL er håndtering av flere samtidige brukertilkoblinger fra en klient.

Ulempen med bruk av PostgreSQL er at enkelte oppgaver enten bør utføres interaktivt eller alternativt via komplekse og tidkrevende skript. Tidkrevende blir det når datamengden blir stor. Disse oppgavene jeg referer til er indeksering av romlige objekter og tilhørende egenskapsdata, men jeg kan også nevne behovet for administrasjon for å endre på relasjoner, noe som for eksempel er nødvendig for å tilføre det ekstra temporale attributtet.
Tidkrevende er et relativt begrep i denne sammenhengen. Med tidkrevende mener jeg tiden målt med brukerens persepsjon ved bruk av systemet.
På bakgrunn av denne persepsjonen har jeg foreløpig trukket konklusjonen at PostgreSQL bør benyttes til lagring av nedbørdata av mer permanent karakter, data som det ofte spørres etter, samt data som fordrer fleksibel datautvelgelse utover det som er mulig i GRASS.
Bruken av PostgreSQL er altså en måte å komme rundt problemstillinger som oppstår pga av ”mangler” i GRASS til å håndtere temporal informasjon, samtidige brukertilkoblinger og sist men ikke minst fleksibel datautvelgelse.

3.5 Apache vevtjener

Apache HTTP tjener er den mest benyttede vevtjener i dag. Apache vevtjener vist seg å ha få sikkerhetsrelaterte problemer.
Tjeneren kan installeres med en rekke valgfrie moduler. Disse gjør det mulig å tilpasse tjenerens funksjonalitet etter behov.
Funksjonaliteten styres av et ekstensivt sett med direktiver. Direktivene forteller Apache hvordan spørringer fra klienter skal håndteres og hvordan serveren skal fungere. Det er også mulig å legge til nye direktiver med ny funksjonalitet ved hjelp av statisk eller dynamisk lenkede programtillegg.

Figur 16: Vevtjener og klient kommunikasjon



Kartdatautvelgelse sett fra klientapplikasjonen skjer via en vevtjener med HTTP (hypertext transfer protocol) protokollen og CGI (Common Gateway Interface).
HTTP er en tilstandsløs protokoll. Etter at en spørring er utført og respons er sendt fra tjeneren til klienten, lukkes nettverksforbindelsen. Ingen tilstandsinformasjon lagres mellom hver interaksjon.
I dette systemet er vevtjenerens funksjon å håndtere forespørsler og returnere kartdata på vektorformatet SVG eller alternativt X3D.
Alternative raster- og vektorformater er mulig som følge at Apache vevtjener er en applikasjon som håndterer alle mulige forespørsler via HTTP protokollen.
Vevtjeneren håndterer formatering av dokumenter som svar på forespørsler enten internt, via programtillegg eller CGI skript. Responsen sendes til klientapplikasjonen som typisk er en nettleser.

Klientapplikasjonen adresserer en forespørsel med en URL (Uniform Resource Locator).
En URL inneholder informasjon om hvordan og hvor en ressurs skal gjøres tilgjengelig. Den inneholder informasjon om hvilken protokoll som skal benyttes, adressen til hvor tjeneren befinner seg på nettet, hvor ressursen er lagret og opsjonell informasjon som er nødvendig for å kunne få tilgang til ressursen.
En URL er typisk bygget opp på følgende måte:

<protokoll>://<server>/<ressurs><opsjon(er)>

En ressurs kan uttrykkes eksplisitt eller baseres på implisitt tolkning internt i vevtjeneren.

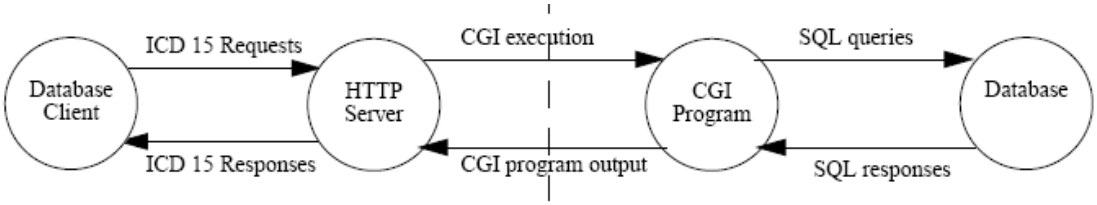
3.6 Prosessering på tjenersiden

Prosessering på tjenersiden via en vevtjener realiseres ofte som såkalte CGI applikasjoner.
CGI (Common Gateway Interface) er et grensesnitt hvor klientapplikasjonen (typisk en nettleser) kan få tilgang til applikasjoner via vevtjeneren. CGI applikasjoner kan være utviklet i alle mulige programmeringsspråk, fra enkle skallskript til kompilerende språk som produserer binær eksekverbar programkode. Ofte er slike applikasjoner utviklet i skriptspråk som for eksempel Perl. Fordi skriptspråk ofte er benyttet til å utvikle CGI applikasjoner, ser man også termen ”CGI skript” brukt nærmest synonymt.
CGI grensesnittet definerer også metoder for å overføre programvareparametere.

En typisk CGI applikasjon vil kanskje lese data fra en eller annen kilde (for eksempel en database), dernest formatere data på et format som klientapplikasjonen kan tolke. Formaterte

data fra en CGI applikasjon skrives på standard tjenerutgang som vevtjeneren leser og returnerer data emballert i HTTP protokollen til klientapplikasjonen.

Figur 17: CGI databaseapplikasjon [http://www.gemini.edu/documentation/webdocs/icd/icd_15.pdf]



En CGI applikasjon kan også returnere data direkte til klientapplikasjonen, men slike applikasjoner øker kompleksiteten siden de må skrive ut http kontrollhoder selv. Kontrollhoder er informasjon som sendes i respons på en forespørsel før selve dokumentet.

Eksempel:

```
HTTP/1.0 200 OK

Server: Netscape-Communications/1.12

Date: Monday, 21-Oct-96 01:42:01 GMT

Last-modified: Thursday, 10-Oct-96 17:17:20 GMT

Content-length: 1048

Content-type: text/html

CrLf
```

Slike applikasjoner omtales som nph applikasjoner (no parsing of headers). Applikasjonen må ta oppgaven med å skrive ut HTTP kontrollhoder, noe som normalt utføres av vevtjeneren. Fordelen er at man optimaliserer med hensyn på responstiden, men utført galt vil klientapplikasjonen gi en feilmelding når den mottar dokumentet. I noen vevtjener realiseres en slik metode kun med eksplisitt å endre innstillinger i oppsettsfilene.
For at vevtjeneren skal vite at det er en nph applikasjon, benytter man seg av en navnekonvensjon med ”nph-” prefiksing av navnet på applikasjonen.

Systemarkitekturen i Figur 15 angir lagring av nedbørdata i PostgreSQL og GRASS. CGI applikasjoner for PostgreSQL er nærmest trivielt å utvikle. Som nevnt tidligere så har eksempelvis Perl et generell database-grensesnitt kalt DBI (Database Interface) som er et abstraksjonslag mot forskjellige databasedrivere. Andre programmeringsspråk har liknende løsninger for PostgreSQL. I så måte bør man også nevne PHP (se kapittel 1) som er et interpretspråk man kan bygge inn i for eksempel SVG eller HTML dokumenter, men som prosesseres på tjenersiden. PHP er ofte brukt for å utvikle prosesser mot databaser.

Hva med GRASS brukergrensesnitt?
Applikasjoner kan få tilgang til GRASS gjennom å sette verdier til GRASS miljøvariable.

Applikasjonene gis da tilgang til GRASS miljøet og kan automatisere produksjonen av karttema for nedbørdata. Dette er beskrevet i kapittelet som omhandler GRASS.
Problemet er som tidligere nevnt samtidige brukertilkoblinger.

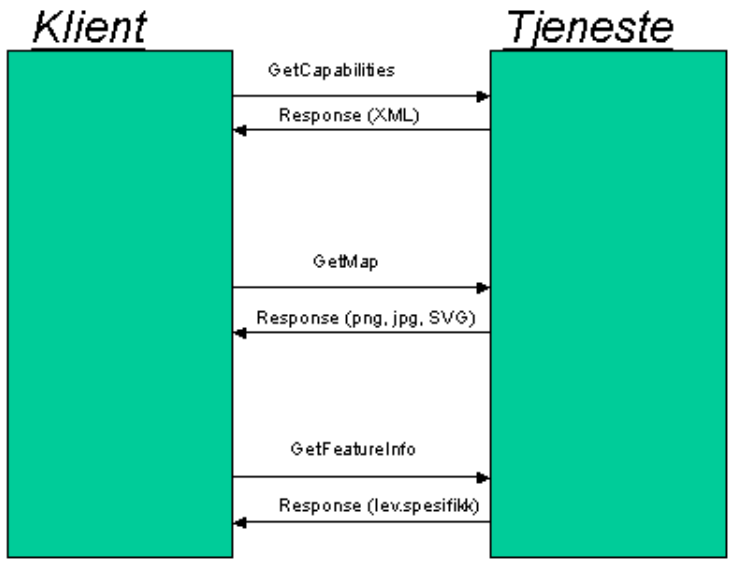
Et annet alternativ er UMN MapServer applikasjonen som nesten fortjener et eget kapittel.
Se kapittel 1 for en kort beskrivelse av MapServer.
MapServer kan sees på som en stor og kompleks CGI applikasjon for å håndtere spørringer mot flere kilder med romlige data, og returnere kartbilder på veldig mange formater, deriblant SVG, men ikke X3D. MapServer støtter både raster- og vektorformat.
MapServer kan foreta romlige spørringer mot både GRASS og PostgreSQL/PostGIS

I tillegg kan MapServer lage overlagskart (kart hvor flere karttema er lagt oppe på hverandre, overlapping), gi ut målestokk, referansekart og tegnforklaringer for å nevne noe.
MapServer kan også formatere karttema på Macromedia Flash format.

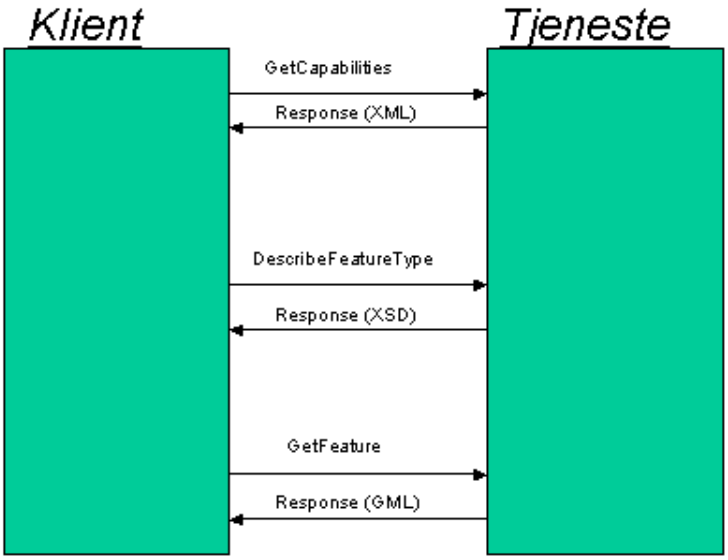
Oppsett av MapServer gjøres med bruk av mal- eller sjablongfiler. Malen som ønskes brukt overføres som en parameter til MapServer.
MapServer støtter også flere programutviklingsmiljøer via et applikasjonsgrensesnitt (eng. API) kalt Mapscript.
PHP/Mapscript er en egen modul som støtter utvikling i PHP.
Foruten PHP støtter MapServer utvikling i programmeringsspråkene Perl, Java, Python og C#.
MapServer kan settes opp til å fungere som WMS/WFS-klient eller tjener. WMS- og WFS-tjenere kan også implementeres som egne tjenerprosesser som er utviklet for å håndtere forespørsler.
WMS formaterer returdokumenter i PNG, GIF eller JPEG rasterformat, men støtter også vektorformatene SVG eller WebCGM (Web Computer Graphics Metafile). Se Figur 18.

WFS er en ren kartdatatjeneste med tekstkoding av data i GML (se kapittel 1). GML er et XML markeringsspråk i likhet med SVG, og det er derfor mulig å transformere GML kode til SVG via XSLT (se kapittel 1og Figur 19).

Figur 18: WMS prinsippskisse
[http://www.geodata.no/upload/Konferanser/ForedragBK2006/WMS%20og%20WFS.ppt]



Figur 19: WFS prinsippskisse
[http://www.geodata.no/upload/Konferanser/ForedragBK2006/WMS%20og%20WFS.ppt]



3.7 Prosessering på klientsiden

I dette avsnittet drøftes prosessering på klientsiden ut fra forutsetningen at klientapplikasjonen er en nettleser.
Prosessering i nettleser er enten realisert som en del av nettleseren eller at nettleseren støtter programtillegg som en måte å utvide sin funksjonalitet på.

Vi har skiller ofte mellom tre typer klienter i et nettverk, nemlig tynne, tykke og hybride klienter. Se tabellen som gir en oversikt over forskjellen på klienttyper.

En fet (tykk, rik) klient foretar størsteparten av all prosessering selv og er ikke avhengig av en tjener for å utføre denne prosesseringen. Typiske klienter er PC (Personal Computer) og laptopper eller bærbare PC-er som det også kalles.

Tynnklient er en minimalistisk klienttype som baserer seg på de ressursene en tjener kan stille til rådighet. I denne sammenhengen benyttes ofte ordet applikasjonstjener. Tynnklienten er ofte bare et grafisk brukergrensesnitt som visualiserer responsen fra applikasjonstjeneren. All prosessering og ressursutnyttelse skjer på tjeneren.

Hybridklient er da en blanding av tykk- og tynnklient. Den kan prosessere lokalt, men er avhengig av tjener for lagring av data. Denne relativt nye tilnærmingen tilbyr egenskaper som multimedia, høy ytelse fra tykk-klienten, og oversiktlig administrasjon og fleksibilitet fra tynnklienten.

Drøftelsen er tatt med for å illustrere hvilke problemstillinger som er relevant i tilknytning til vevbasert GIS arkitektur. ”Hjemmeklienter” tilkoblet Internett er vanligvis en laptopp eller PC. Bedriftsklienter tilkoblet intranett og Internett er nødvendigvis ikke bare tykk-klienter. Det er ofte en blanding av tykk- og tynnklienter alt etter hva en bedrift legger opp til.

Systemarkitekturen vil i prinsippet kunne støtte alle typer klienter, men hvor godt er et spørsmål om infrastruktur og maskinressurser, og er problemstillinger en ”hjemmebruker”, eller en bedrift og dens brukere må ta stilling til.

Prosessering på klientsiden skjer ved hjelp nettleseren eller ved bruk av et programtillegg. Programtillegg (eng. plugins) er en utvidelsesarkitektur som støttes av de mest populære nettlesere for å utvide funksjonaliteten.
Formatet på dokumentene som returneres, er basert på åpne de facto standarder definert i W3C og OGC.
Det forventes at nettlesere støtter flere av disse standardene fullt ut. Sannheten er dessverre noe broket. Flere av nettleserne støtter flere av de åpne standardene, men som oftest kun et subsett av originalstandarden.

World Wide Web Consortium (W3C) har vedtatt standarder for tolkning av web-basert innhold.
Ved å utgi nettlesere som ikke konsekvent støtter disse standardene, skader nettleser-produsentene både webutviklere, næringer og brukere.
Mangel på enhetlig støtte for W3C standarder gjør bruk og utvikling av web-baserte teknologier unødvendig vanskelig og dyrt.
Vi er inneforstått med behovet for innovasjon i et marked i rask utvikling. Men grunnleggende støtte for eksisterende W3C standarder har blitt ofret i innovasjonens navn, noe som har ført til unødig fragmentering av WWW og redusert tilgjengelighet.
Vårt mål er å støtte disse grunnleggende standardene og å oppfordre utviklerne av nettlesere til å gjøre det samme, og dermed sikre en enkel, overkommelig tilgang til webteknologier for alle [http://archive.webstandards.org/missionno.htm oversettelse av: Tomas Fjetland].

Relevante åpne standarder som bør støttes i en nettleser er:

Strukturerte språk:

XML
HTML

Tilleggsbehov i systemarkitekturen:

SVG
GML
VRML/X3D

Presentasjonsspråk:

CSS1- og 2.

Objektmodeller:

DOM - Document Object Model 1 Core HTML/XML

Skriptspråk:

ECMA-skript (standardisert Javaskript)

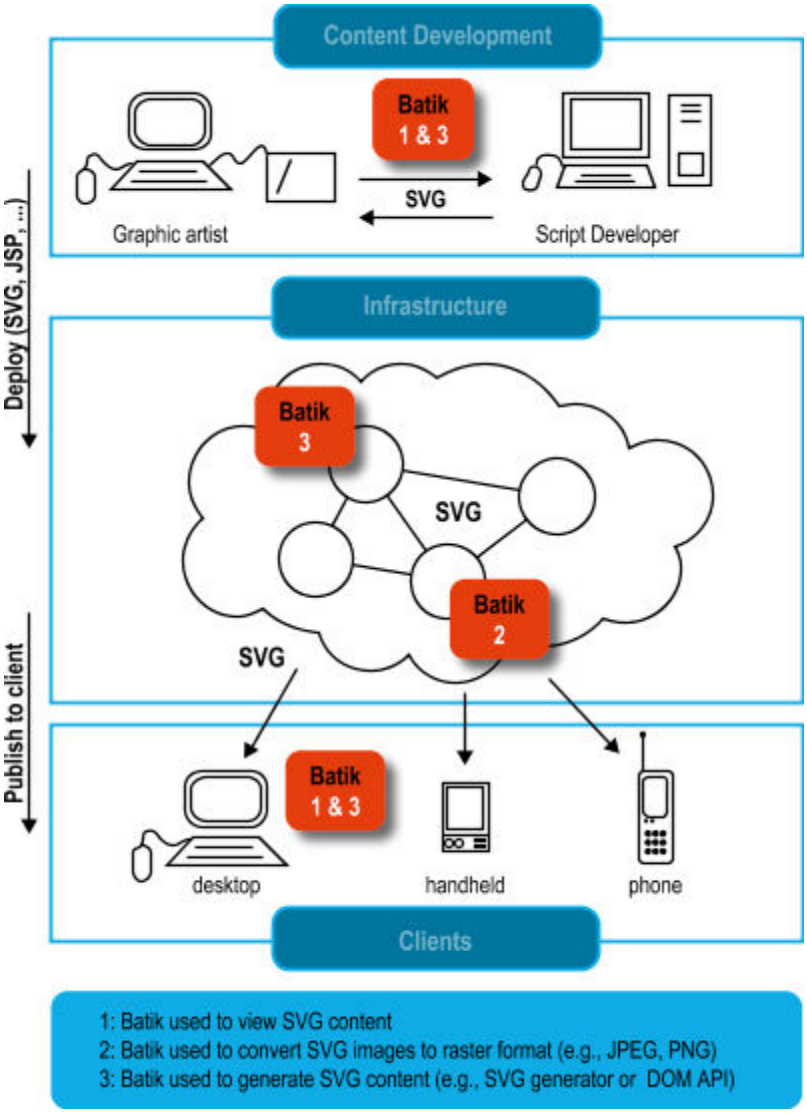
Hvordan håndtere subsett implementeringer av standarder?

Det syntes for meg at en fornuftig praksis vil være konsekvent bruk av programtillegg. Å utvikle med henblikk på bruk av programtillegg vil si at behovet for ”digresjoner” i koden faller bort, samt at administrasjon av koden er mer oversiktlig.

Batik er et Java utviklingsverktøy for å generere grafikk på SVG format i frittstående applikasjoner eller Applets (se http://xmlgraphics.apache.org/batik/). Batik er et XML grafikkprosjekt fra Apache Software Foundation.

Prosjektets ambisjon er å gi utviklere et sett med kjernemoduler som kan brukes sammen eller hver for seg til å utvikle SVG løsninger. Eksempler på moduler er ”SVG Parser”, ”SVG Generator” og ”SVG DOM”. Prosjektets ambisjon er også at Batik skal være utvidbar. Batik tillater utviklere å takle kundespesifiserte SVG elementer.
Selv om målsettingen er å levere et sett med kjernemoduler, så medfølger en komplett SVG leser, en implementasjon som brukes til å verifisere de enkelte moduler og deres interoperabilitet.
Batik har også metoder for å konvertere SVG til flere rasterformater og andre vektorformater.
Batik støtter et subsett av SVG standarden. Animering er blant annet ikke støttet. Se http://xmlgraphics.apache.org/batik/status.html for nåværende utviklingsstatus i henhold til SVG standarden.

Figur 20: Eksempel på bruk av Batik [http://xmlgraphics.apache.org/batik/]



”Adobe SVG Viewer” er et programtillegg for å tolke dokumenter på SVG format. Den støtter ikke SVG standarden 100 %, men er den SVG motoren som har best støtte for standarden blant eksisterende løsninger. Se http://www.adobe.com/svg/indepth/pdfs/CurrentSupport.pdf for dokumentasjon og status for konformitet med SVG standarden.
”Adobe SVG Viewer” har enkelte funksjonelle problemer mot noen nettlesere. Dette er godt dokumentert og problemer ved valg av nettleser hos bruker kan unngås.
”Adobe SVG Viewer” har en innebygd skriptmotor, DOM grensesnitt og i tillegg støtte for formateringsspråket CSS.

Hva med programtillegg for VRML og X3D standardene (se kapittel 1)?

Se http://cic.nist.gov/vrml/vbdetect.html for eksiterende programtillegg og applikasjoner som støtter VRML og X3D. Av disse har jeg vurdert ”Octaga Player” som kan lastes ned som både et programtillegg for nettlesere og som et utviklingsverktøy av VRML og X3D dokumenter.

”Octaga Player” utviklet av Octaga AS (et norsk firma) er et programtillegg utviklet for nettlesere. Implementasjon eksisterer både for Linux og Microsoft Windows operativsystem. Programtillegget er gratis å laste ned og bruke. Dokumentasjonen er god, se http://www.octaga.com/ hvor man kan lese mer om ”Octaga Player” på deres ”skryteside”. De norske bedriftene Statoil og Telenor er blant brukerne. Octaga AS er i skrivende stund blitt medlem av Web3D konsortium.

Foreløpig eksisterer et begrenset antall programtillegg for X3D standarden. Noen flere programtillegg og applikasjoner eksisterer for VRML da standarden har en lengre historikk. Ønsker man å utvikle 3D dokumenter bør man vurdere programtillegg som støtter standarden X3D. Argumentet for en slik strategi er at X3D standarden bygger videre på VRML standarden, og forventes således å kunne lese dokumenter formatert i henhold til VRML standarden. Ekstra arbeid unngås hvis man har eksisterende dokumenter basert på VRML formatet. Alternativt vil man naturligvis utvikle 3D grafikk basert på den nyeste standarden.

3.8 Nettleser

Se forrige avsnitt hvor drøftelser om prosessering på klientsiden inkluderer nettleser i samme problematikk med hensyn på åpne standarder.
Jeg vil kort å kommentere valg av nettleserne Opera og Mozilla Firefox i mitt arbeid. Hensikten med valg av nettlesere har ikke vært å diskriminere andre nettlesere. Mozilla Firefox er valgt for å verifisere at systemarkitekturen støtter nettlesere basert på åpen kildekode. Det er et krav.
Opera er ikke åpen kildekode men gratis å laste ned og bruke. Opera er valgt fordi den har bedre støtte for W3C standardene enn Mozilla Firefox.
Argumentet mitt for å benytte disse to nettleserne er at begge har innebygd støtte for SVG standarden. Begge støtter funksjonell utvidelse med bruk av programtillegget ”Adobe SVG Viewer”. De er også implementert på flere maskin- og OS plattformer.
Begge nettlesere støtter utvikling av dynamiske og interaktive visualiseringer med bruk av Javaskript og DOM.
Jeg har dermed kunnet verifisere at der er minimum to nettlesere en bruker kan benytte kostnadsfritt, og som kan kjøre på flere maskin- og OS plattformer. At nettleserne er populære har også hatt innvirkning på valget. Sannsynligheten for at nettleserne allerede er installert hos bruker og er noe brukeren fortsatt kan bruke er en fordelaktig strategi.

Det er flere måter å verifisere hvor godt en nettleser oppfører seg i forhold til standardene. En kjent nettlesertest er Acid2-testen.
Acid2-testen ser etter grunnleggende støtte og implementasjon av de anbefalte standardene fra W3C i nettlesere, dessverre er det ennå ikke så mange nettlesere som består testen. Hensikten med testen er å få utviklere av nettlesere til å implementere standardene slik at standardkonforme nettsider er kompatible uavhengig av hvilke plattformer og nettlesere de vises på. ”The Web Standards Project” (se også http://www.webstandards.org/) står bak utviklingen av Acid2-testen.
I skrivende stund har Opera versjon 9 har passert denne testen, mens Mozilla Firefox har ikke. Det har heller ei Microsofts Internet Explorer (IE).

3.9 Sluttkommentarer

Systemarkitekturen gir mange opsjoner for dynamisk visualisering av kartobjekter på mange formater og ikke kun SVG, selv om SVG er fokuset i mitt arbeid.
Utvikling av tolkerprogramvare av W3C-standarder bør resultere i konforme løsninger med hensyn på visualisering.
Med konforme løsninger menes at visualisering av kartdokumenter bør ha samme utforming og lik dynamikk.
Strategien for å unngå subsett problematikk bør være å bruke programtillegg. For X3D standarden er bruk av programtillegg den eneste mulige løsningen i dag.

Hvordan utvikle konforme, dynamiske og interaktive grafiske løsninger på SVG format?

Strategien med bruk av programtillegg er en administrativt fordelaktig løsning. Ved kun å utvikle SVG formaterte dokumenter unngår man problemer relatert til valg av nettleser og varierende konformitet med W3C standarder.
Alternativet med ukritisk utvikling av dynamiske vevsider basert på innebygd støtte i nettlesere, vil være å bygge inn ekstra kode som detekterer nettlesertype, og som gir en alternativ løsning og ofte uønskede effekter. Dette bidrar til å øke kompleksiteten i systemadministrasjon og kode.

Andre alternativer er å være selektiv og eksplisitt diskriminere typer av nettlesere, eller enda verre å forårsake degradert visualisering og/eller feilmelding i nettleser hvis man ignorer problemstillingen.

For både utviklere og brukere har man alt å vinne på å satse på konforme løsninger.
Arkitekturen støtter mange mulige løsninger, men viktige avveininger må tas med hensyn på en persepsjon av brukervennlighet.

4 Kart og karttyper

Bruk av rammeverket fordrer basal kartografisk kunnskap. Hva trenger vi å vite?
Vi trenger geografisk informasjon som forteller oss kartobjektets geografiske utstrekning, målestokk og orientering for å nevne noe. I tillegg fordres det kjennskap for å omforme disse verdiene ved interaktive operasjoner i kartdokumentet. Målet er at visualisering av kartdokument hele tiden skal gi korrekt geografisk informasjon. Da må man kjenne til basale beregninger og referanser innen kartografi.

4.1 Kart

Hva er et kart?

Det er mange definisjoner på hva et kart er. De fleste tenker på et kart som en grafisk fremstilling på et papir over et landområde betraktet med et høydeperspektiv som angir en gitt målestokk.

En annen forklaring er at kart er en gjengivelse av terrenget, en projeksjon av terrenget til horisontalplanet.
En mer relevant forklaring i forhold til mitt arbeid er digitale eller analoge (skjerm eller papir) utdata fra et GIS som viser geografisk informasjon ved å bruke etablerte konvensjoner innen kartografi. Et kart er det endelige resultatet av en serie med steg med GIS prosesseringer [Longley, 2005].

Viktige konvensjoner innen kartografi er datum, projeksjon, koordinatsystem og målestokk. Jeg vil prøve å gi en forklaring på hver av disse konvensjonene.

Kartet skal gi oss informasjon, og vanlig er å dele kart i to ulike typer etter hvilket formål kartet skal tjene.

- Topografiske kart er en generell presentasjon av mange synlige detaljer i terrenget
- Tematiske kart fremhever et eller ett fåtall tema. Temakartet kan vise forhold og egenskaper som direkte eller indirekte kan knyttes til objekt i kartet.

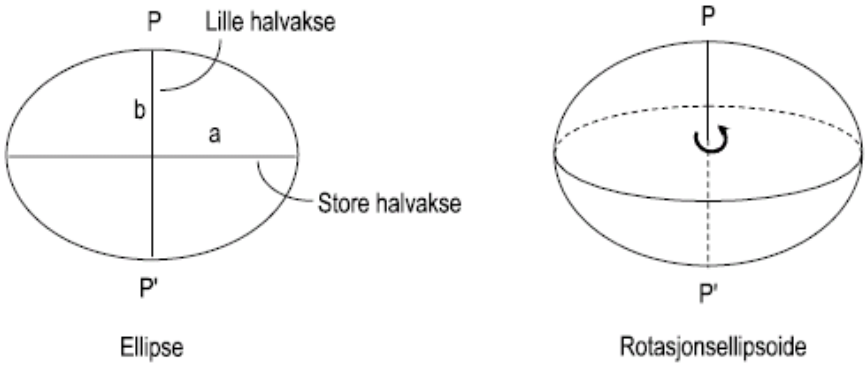
4.2 Datum

Et datum er en numerisk eller geometrisk størrelse eller et sett av slike størrelser som danner utgangspunkt/basis for andre størrelser. Datum er altså en referanse og i GIS snakker vi om en geometrisk referanse, eller mer korrekt en geodesisk referanse. Geodesi er en av de eldste naturvitenskaper og omhandler jordens form, gravitasjon og endring. Et geodetisk datum omfatter vanligvis størrelse og form på en ellipsoide (jordmodell) som benyttes i kart- og oppmålingsvirksomhet, foruten denne ellipsoidens plassering og orientering i forhold til den fysiske jorden. I et geodetisk datum blir alle høyder gitt som ellipsoidiske høyder (høyden over ellipsoiden).
Frem til man startet med satellittbasert geodesi, var et tradisjonelt geodetisk datum knyttet til jordoverflaten (toposentrisk datum). Et toposentrisk geodetisk datum kan være nasjonalt og dekke én nasjon, eller det kan være regionalt og dekker da flere nasjoner. Et toposentrisk geodetisk datum krever flere parametere klarlagt enn det som kreves av et geosentrisk datum, som er knyttet til jordens massesenter. Et toposentrisk geodetisk datum ble knyttet til et fundamentalpunkt, vanligvis et observatorium, hvor loddavvik og geoidehøyde måtte angis [kartverk, 2004].

En ellipsoide er en modell av jorden. Jorden er flatrykt ved polene og beskrives best gjennom en ellipsoide. Det finnes flere jordmodeller i kjølvannet av den historiske utviklingen innen geodesi for å kunne fastslå jordens størrelse. Flere jordmodeller er også laget som følge av teoretisk modellering for å oppnå større regional presisjon.

EPSG (European Petroleum Survey Group) vedlikeholder et datasett med geodesiske parametere og datum som beskriver eksisterende koordinatsystemer og formler for transformasjon mellom koordinatsystemene.
Rollen til EPSG ble overtatt av OGP (International Association of Oil and Gas Producers) i 2005, som nå vedlikeholder EPSG databasen som fortsatt er en referanse. Datasettet er tilgjengelig gjennom en EPSG tallkode f.eks. er EPSG:4326 er datumet WGS84 (World Geodetic System 1984) og EPSG:32611 er UTM (Universal Transverse Mercator) sone 11 Nord, WGS84. Se også http://www.epsg.org/.

Figur 21: Ellipsoiden – jordmodellen [Gjevestad, 2006]



$$f = \frac{a-b}{a} \quad = \textit{flattrykning}$$

WGS84 a = 6378137 f = 1 / 298.257223563

4.3 Kartprojeksjon

En kartprojeksjon er en overføring av den krumme jordoverflaten eller deler av denne til en gjengivelse i planet ved et matematisk formelverk eller geometrisk projeksjon.

… geometriske projeksjoner… har som formål å avbilde en geometri på en todimensjonal flate, slik at det er lettere å vise den fram på et flatt stykke papir eller en dataskjerm. Slike projeksjoner er mye brukt for arkitekt- og konstruksjonstegninger og kart,…[Albregtsen, 2006].

Punkter på ellipsoiden med geodetiske koordinater ”overføres” på en flate som kan foldes ut til et plan og gir punkter med koordinater i kartplanet.
Det er matematisk umulig å brette ut et område på jordoverflaten til et kartplan uten at man får fortegninger.
Det er et ønske at projeksjonen skal bevare arealer, vinkler og avstander slik at disse stemmer overens med tilsvarende størrelser målt på jordoverflaten. Det er umulig å oppfylle alle kravene samtidig, men hvis man tar en liten del av jordoverflaten om gangen blir ikke fortegningene så store.
Man kan dele inn kartprojeksjoner på mange forskjellige måter, etter projeksjonsmetode (ekte, uekte), etter egenskaper (vinkeltro, lengdetro, flatetro) eller art. En vanlig grovinndeling etter art vil være: sylinderprojeksjon, kjegleprojeksjon og planprojeksjon, avhengig av om ellipsoiden projiseres inn på en sylinder, en kjegle eller et plan.

Eksempler på projeksjoner kan være:
Sylinder:

- Mercator
- Transversal Mercator (også kalt Gauss-Krüger-projeksjon eller bare Gauss-projeksjon)

Kjegle:

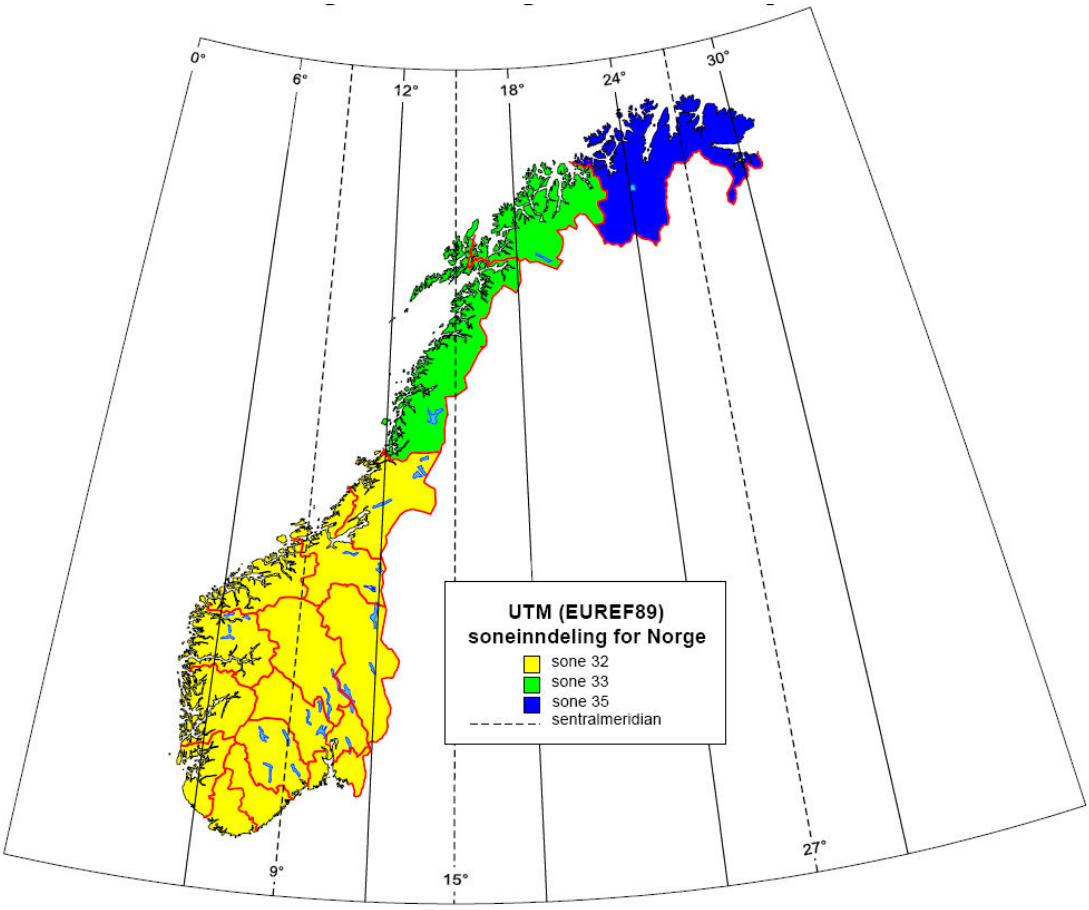
- Polykonisk
- Lamberts konforme kjegleprojeksjon

Plan:

- Stereografisk
- Gnomonisk

En spesiell variant av transversal Mercator-projeksjon går under betegnelsen Universal Transversal Mercator (UTM). Punktene på ellipsoiden konverteres da først til en sylinder som tangerer ellipsoiden langs en meridian. Så tenkes sylinderen brettet ut til et kartplan. Ved utbrettingen benyttes en målestokksfaktor på 0.9996 (det vil si -400 ppm.) både for koordinatverdier langs sentralmeridianen (tangeringsmeridianen) som blir nord-akse og for koordinatverdier langs aksen vinkelrett på sentralmeridianen, det vil si øst-aksen. Selve sentralmeridianen blir gitt en kunstig østverdi på 500 000 for å hindre at østverdier i kartplanet blir negative vestenfor sentralmeridianen. Siden målestokksfortegningen i kartplanet øker med avstanden fra sentralmeridianen, kan man bare anvende projeksjonen ut til en viss avstand fra sentralmeridianen før fortegningen overskrider en fastsatt grense. Da må man legge en ny sylinder langs en ny sentralmeridian og foreta en tilsvarende utbretting. Ved hver slik utbretting dekker man en sone. I UTM er sonene jevnt over standardisert til å dekke 6 lengdegrader, det vil si 3 grader ut til hver side av sentralmeridianen. NATO, som hovedaktør med UTM-systemet, har imidlertid akseptert at sone 32 er utvidet vestover mellom 56°N og 64°N slik at Syd-Norge og det nærmeste havområdet utenfor kommer med i sone 32. Ellers er det få steder UTM-sonebredden avviker fra de nevnte 6 lengdegradene. UTM-sonene går sammenhengende fra 80° sydlig til 84° nordlig bredde.

Figur 22: UTM soner i Norge [Geodesi, 2004]



4.4 Koordinatsystem

Et koordinatsystem er et sett av matematiske regler som spesifiserer hvordan punkt på en linje, i et plan eller i rommet kan tilordnes koordinater. For geodetisk datum og vertikalt datum vil koordinatsystemet være knyttet til jorden. Store deler av teksten og illustrasjoner er hentet fra [kartverk, 2004].

Romlige koordinater (3-dimensjonale):

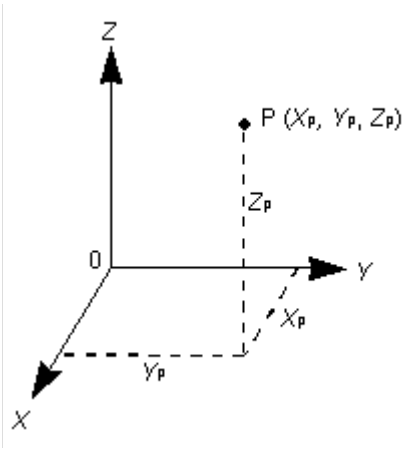
- Geosentriske koordinater (X, Y, Z) er rettvinklede, kartesiske koordinater med origo i jordens massesenter.
- Geodetiske koordinater (φ, λ, h), der φ og λ refererer seg til ellipsoiden og h er ellipsoidisk høyde.
- Geodetiske koordinater (φ, λ) horisontalt og ortometrisk høyde eller normalhøyde (H) vertikalt.

Flatekoordinater (2-dimensjonale):

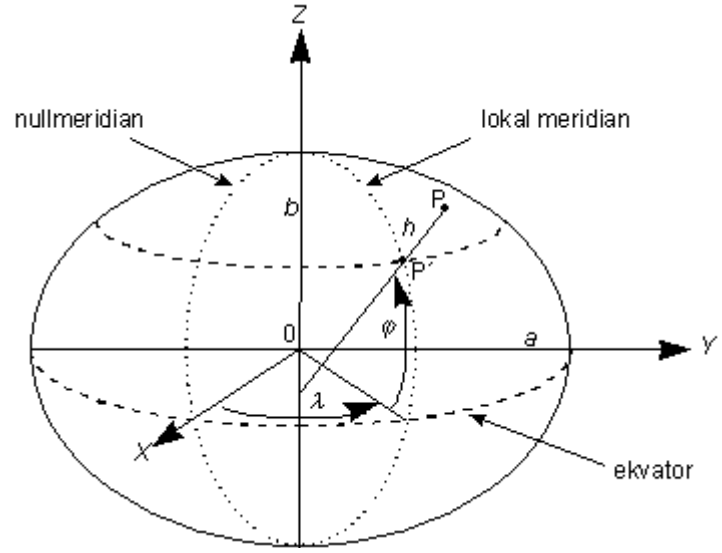
Ved geodetiske koordinater er bredde og lengde gitt i grader (eventuelt også bueminutter og buesekunder) med referanse til en ellipsoide som er tilpasset jorden.

Ofte blir betegnelsen geografiske koordinater brukt, men dette er en samlebetegnelse for geodetiske koordinater (på ellipsoiden) og astronomisk bestemte koordinater (beregnet i forhold til loddlinjen, altså geoiderelatert[4]).
Kartplankoordinater er plane koordinater gitt i kartets projeksjon. Man kan ta ut kartplankoordinater fra et digitalt kart eller fra rutenettet på et analogt kart.

Figur 23: Kartesisk koordinatsystem 3 dimensjoner [Geodesi, 2004].



Figur 24: Ellipsoidiske koordinater og kartesiske koordinater [Geodesi, 2004]



Den matematiske sammenhengen mellom kartplankoordinater og ellipsoidiske koordinatsystem kan matematisk formuleres på følgende matriseform:

[4] **geoiden,** en matematisk nivåflate for jorden. Dannes av verdenshavenes overflate og dens tenkte fortsettelse under landmassene. Avviker neppe noe sted mer enn 100 m fra den rotasjonsellipsoide som oppstår når en lengdesirkel roterer rundt jordaksen. Flaten buler litt opp over kontinentene.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} (N+h)\cos\varphi\cos\lambda \\ (N+h)\cos\varphi\sin\lambda \\ \left(N(1-e^2)+h\right)\sin\varphi \end{bmatrix}$$

Her er *N* ellipsoidens perpendikulærkrumningsradius, også kalt normalkrumningsradius, som er krumningsradien til det normalsnittet på ellipsoiden som står vinkelrett på meridiansnittets plan. Størrelsen *e* er eksentrisiteten for ellipsoiden, og *h* er høyde over ellipsoiden målt langs ellipsoidenormalen gjennom punktet P. Hvis P ligger innenfor ellipsoideflaten, blir *h* negativ.

$e = \sqrt{2f - f^2}$ hvor $f = (a-b)/a$

$N = a/\sqrt{1 - e^2 \sin^2\varphi}$

f er flattrykningen til ellipsoiden, mens *a* og *b* er ellipsoidens henholdsvis store og lille halvakse.

$\rho = \sqrt{X^2 + Y^2}$ $tg\theta = Z/[\rho\cdot(1-f)]$

$\varphi = arctan[\{Z + [e^2/(1-f)]\cdot a\cdot \sin^3\theta\}/(\rho - e^2\cdot a\cdot\cos^3\theta)]$

$\lambda = arctan[Y/X]$

$h = \rho/\cos\varphi - N$

Dette formelverket holder en beregningsnøyaktighet for **φ, λ, h** bedre enn en millimeter ved alle aktuelle høyder relatert til måling på jorden, det vil si høyder med tallverdi mindre enn 10 000 m. Formelen for *h* gjelder ikke ved polpunktet hvor **φ** er 90^o.

Målestokken uttrykker et forhold mellom avstander på kartet og avstander i terrenget. Dersom en kartet har målestokk 1:50 betyr det at 1 lengdenhet på modellen er femti lengdeenheter i virkeligheten. På et kart med målestokk 1:10000 er 1cm på kartet lik 10 000 cm = 100m i terrenget. Vi bruker deletegn for å uttrykke målestokken. Målestokker over 1:250000 er små målestokker. Midlere målestokk er kart i området1:25000 til 1:100000, mens stor målestokk er alt mindre 1:25000, typisk økonomiske kart eller plankart [kartverk, 2004].

MI har i dag flere typer kartdata tigjengelig.

Kjøpte produkter (underlagt lisensbestemmelser):

- Norge i målestokk 1:250000 (N250) fra Statens Kartverk.
- Norge i målestokk 1:2000000 (N2000) fra Statens Kartverk.
- Hele verden i målestokk 1: 1000000, (Digital Chart of the World) fra ESRI.

Andre/avledete data.

DTM (Digitale terrengmodeller):

- 1 x 1 km - Fra Statens Kartverk.
- 100 x 100 m - avledet fra N250, nøyaktighet +/- 100 m .

5 Vektorformat og geometrisk modellering

Nedbørdata som punktdata med desimaloppløsnning gir liten mening fordi disse dataene allerede er beheftet med en usikkerhet som er større enn en tiendedels millimeter. Klimadivisjonen bruker å dele inn nedbørdata i intervaller som kan variere med størrelsen på geografisk område som visualiseres eller tidsperioden (måned, år). Ved å generalisere reduserer vi antall distinkte verdier for nedbørmengde, og ved å omforme dette til vektorformat reduserer vi også behovet for geometrisk informasjon.

Nedbørdata på vektorformat er også nødvendig når vi skal omforme nedbørdata til interaktive kartdokument hvor vi kan manipulere geometrien, dvs. operasjoner som zooming og valg av fargeattributter for kort å nevne noen operasjoner.
Vektorformatet har en rekke fordeler i forhold til rasterdata ved visualisering. Grafikk på vektorformat er god og nøyaktig og beholder disse egenskapene med endring av grafikken som forstørring/forminsking.

Generalisering av punktdata til andre geometrisk figurer som polygon, linjer trekanter etc. er det som kalles geometrisk modellering. Hvilke geometriske objekter ønsker vi som geometrisk representasjon av nedbørdata?

Det er også viktig at representasjonen er korrekt! Hvis for eksempel et sett med ruter i et rutenett representerer et geografisk område, vil vektorisering og geometrisk modellering representere den samme geografiske utstrekningen?
Har vi på noen måte forverret kvaliteten og kvantiteten på nedbørdata?

Vektorisering og geometrisk modellering er temaet i dette kapittelet.

En vektor (fra latin *ve'ctor* latin ”bærer”; avledet av det latinske verbet *vehere*, ”føre, bære”) er et matematisk objekt som har både størrelse og retning.

Prosessen med å omforme rasterdata til vektorformat kalles vektorisering. Rasterdata blir omformet til de geometriske formene punkt, linjer, polygoner og volumer for å beskrive formen og beliggenheten til fenomenet nedbør. Vi sier at virkeligheten består av fenomener, og modellen av objekter. I GIS kaller vi modellene for geografiske objekter. Geografiske objekter kan ha forskjellige geometriske verdier.

Et annet viktig begrep er topologi. Vektortopologi beskriver hvordan geometriske verdier ligger i forhold til hverandre. Denne informasjonen kan vanligvis utledes av den geometriske representasjonen. I geografiske databaser er det vanlig å lagre topologisk informasjon.

Et **punkt** lagres som koordinater x, y i 2D og x, y, z i 3D. Et punkt kan være en geometrisk representasjon for et geografisk objekt - f. eks en målestasjon - eller et singulært fenomen som klimaparameter fra en målestasjon.

Ei **linje** lagres gjerne som en punktsekvens. Hvert punkt i sekvensen skal ligge på den linja vi ønsker å representere, men mellom punktene må vi gjøre antagelser om hvor linja går. Det er vanlig å trekke rette linjer mellom punktene i sekvensen når en tegner ut linja.
Dersom vi har for stor avstand mellom punktene på linja (lav samplingsrate) i forhold til målestokken på det ferdige kartet, vil linja da se hakkete ut når vi tegner den ut.

Det er også mulig å tegne ut ei glatt kurve som går gjennom punktene. Det er mange måter å gjøre dette på. Linjer kan også lagres som parametriske funksjoner (f.eks. Bezier og B-spline). Denne metoden vil kunne gi glatte kurver uten knekkpunkter. Parametriske funksjoner kan være nyttige når vi skal representere f.eks. kurver på veiobjekter.

En **polygon** (areal) lagres som et lukket sett med grenselinjer. I tillegg kan det lagres et punkt assosiert med flaten kalt sentroiden. I en konveksformet flate ligger alltid sentroiden inne i flaten. For konkavformede flater kan den ligge utenfor. For sirkelformede flater ligger den i sentrum. I 3D representerer sentroiden massesenteret.

Hva ligger i begrepet geometrisk modellering?

En definisjon:

- *En geometrisk modell beskriver egenskaper til geografiske objekter som form, størrelse, utstrekning, posisjon, lengde og bredde.*

Begrepet geometrisk modellering kom først i bruk sist i 1960- og i begynnelsen av 1970 årene, en tid hvor det skjedde en rask utvikling i PC grafikk og CAD (computer aided design) og produksjonsteknologi. Geometrisk modellering beskriver en samling relaterte matematiske metoder som er noe løst integrert, men kan benyttes til å beskrive formen til et objekt eller uttrykke en fysisk prosess ved å benytte en geometrisk metafor.

Virkeligheten er meget kompleks og inneholder en uendelighet av variasjon. For å gjøre det mulig for oss å forstå virkeligheten eller beskrive den i en datamaskin, benyttes modeller. En modell av virkeligheten er ikke identisk med virkeligheten, men uttrykker bare visse egenskaper og forhold ved virkeligheten. Et kart, for eksempel, er en modell av objekter og fenomener som har en romlig utbredelse [Bjørke, 2005].

En GIS-løsning må støtte lagring av kartdata på både raster- og vektorformat med henblikk på geometrisk modellering. Kompleksiteten i algoritmer for geometrisk modellering er bestemt av kartformatet. Visse algoritmer utføres raskere når formatet er raster, mens andre igjen trekker fordeler av at kartdata er på vektorformat.

Satt i kontekst med nedbørdata – hvordan modellere nedbørdata – hva ønskes visualisert og hvilken informasjon skal presenteres?

Det er ikke en umiddelbar fasit svar på spørsmålet. Se Figur 1 for hva Klimadivisjonen mener er en hensiktsmessig visualisering.

Før vi starter med å drøfte vektorisering av rasterdata, så er det hensiktsmessig å bestemme hvilke typer nedbørkart som ønskes laget. Det er helt sikkert mange måter å modellere nedbørdata på. Vektorisering og geometrisk modellering i GRASS (se neste kapittel) danner datagrunnlaget for visualisering. Jeg vil i dette kapittelet drøfte ulike flatemodeller samt høydekurver (konturer) som er relevante, og konvensjonelle metoder for modellering av nedbørdata.
Geometrisk modellering benyttes til analyse av nedbørdata. Relevante og mulige problemstillinger vil bli drøftet.

5.1 Flatekart - polygoner

En polygon er en ”lukket” sekvens av punkter og linjer, og beskriver et areal.
Prosessen kalles polygonisering eller flatedanning, dvs. en mekanisme som bygger opp flatestrukturer.

Det finnes et utall algoritmer for å ekstrahere polygoner. Å ekstrahere et areal/polygon fra rasterdata er ofte basert på en eller annen form for linjesporingsalgoritme. Prinsippet er å følge randen eller grenselinjene til et areal fra start og tilbake til et utgangspunktet. Arealet innenfor grenselinjene beskriver da rasterdata med ”like” verdier som kvalifiserer dem til å være definert i et gitt areal. Dette er et veldig enkelt prinsipp, men implisitt ligger mange komplekse problemstillinger. For eksempel hvordan behandle raster data som pga av sine verdier danner en ”øy” inne i et areal?
Hvordan tegne grenselinjer når kun hjørnepunkter berører hverandre mellom to rasterdata?
Den mest elementære problemstillingen er å kanskje hvordan få til jevne eller rette grenselinjer?

Jeg vil her gi et eksempel på en algoritme kalt ”kvadrat/rutenett”-algoritmen, som er veldig basal, og er en av de første linjesporingsalgoritmene. Hensikten med å drøfte en slik algoritme er å illustrere problemstillingene man støter på, og som mer komplekse algoritmer bør håndtere.
Knippet med tekst og illustrasjoner har jeg i utgangspunktet hentet fra http://www.cs.mcgill.ca/~aghnei/index.html. Jeg har oversatt teksten fra engelsk, og omarbeidet innholdet noe, slik at innholdet samsvarer mer til problemstillingen i oppgaven.

Algoritmen begynner med å definere et startpunkt for grenselinjegjenkjenning. Et startpunkt kan for eksempel finnes med å skanne alle kolonner i et raster fra ”topp til bunn” eller omvendt for å finne et punkt som møter kriteriet for å være et randpunkt i et areal. Figur 25 viser et raster med edderkoppen plassert i et slikt startpunkt.
Enkelt sagt vil denne algoritmen gjenkjenne alle ruter som grenser opp mot ruter som omslutter et areal av ruter med ”samme” verdi.

Følgende tekst og pseudokode er en mer formell beskrivelse av algoritmen.

Innverdi: Et raster **R** bestående av en komponent **A** av blåfargede ruter som grenser opp mot hverandre og danner et sammenhengende raster.

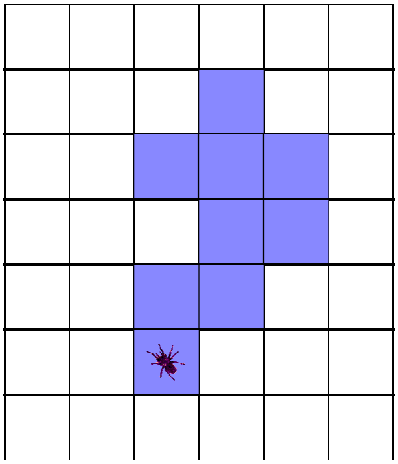
Utverdi: En sekvens av ruter G**(g_1,g_2,g_3…, gn)** som omgir komponent **A** i **R**.

Start på algoritmen:

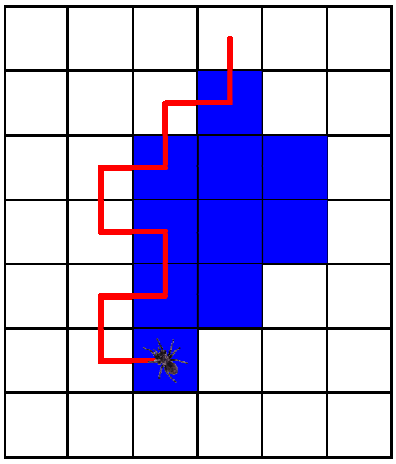
Sett **G** til å være den tomme mengden av ruter.
Fra bunn til topp (eller omvendt) og fra venstre til høyre skann ruter i **R** inntil en blå rute, **s**, i **A** er funnet.
Sett inn **s** i **G**.
Sett nåværende rute **p**, til å være startruten, **s**.
Drei til venstre og entre naboruten til **p**.
Oppdater **p** dvs. nåværende ruteposisjon er nå **p**.
Fordi **p** ikke er **s** utfør:
    Hvis ruten i nåværende ruteposisjon er blå

Sett inn **p** i **G** og drei venstre til neste naborute.
Oppdater **p** dvs. **p** er nåværende ruteposisjon.
Ellers
Gå til høyre naborute.
Oppdater **p**.
Slutt Fordi
Slutt

Figur 25: Startpunktet på algoritmen.

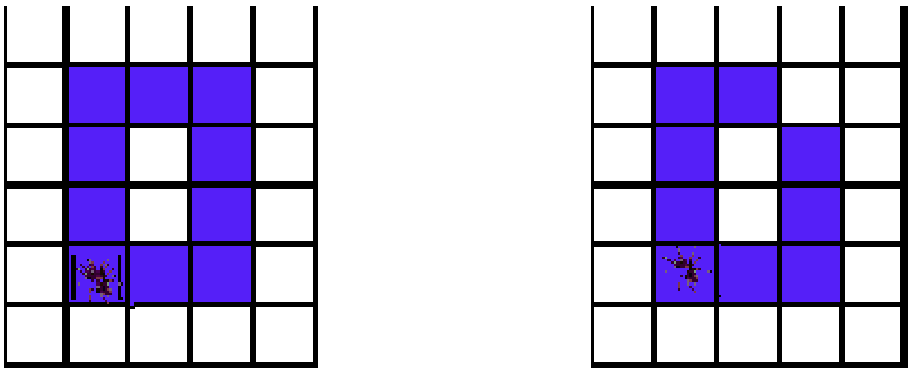


Figur 26: Edderkoppens algoritmesti.



Det er flere og ganske åpenbare problemstillinger en firkantalgoritme må håndtere. Hvilken sti vil edderkoppen følge i Figur 27?

Figur 27: To problemstillinger med firkantalgoritmen.



Det er ikke meningen å gi en fullstendig redegjørelse av algoritmen. Se oppgitte referanse I innledningen av dette avsnittet for mer detaljer.
Spørsmålet en kan stille seg er om algoritmen er anvendelig for praktiske formål?
Det viser seg at algoritmen fungerer feilfritt på en spesiell familie av rutemønstre.

Etter at algoritmen er kjørt sitter vi igjen med en sekvens av ruter, som er et utgangspunkt for vektorisering. Vi må bestemme hvordan en grenselinje skal representeres. Det enkleste ville vært å betrakte midtpunktet i et raster som en punktverdi og deretter tegne opp vektorlinjer mellom hvert punkt og lagre dette. En ytterligere forbedring betyr at man er i stand til å gjenkjenne at flere punkter ligger på en linje, slik at man kun lagrer endepunktene for linjen – reduksjon av datamengden.

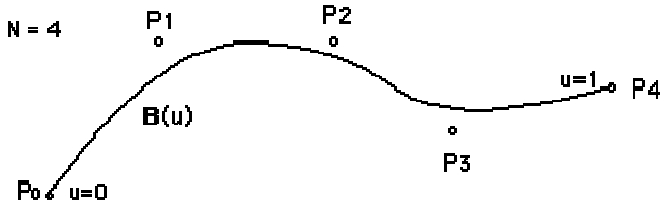
Et bidrag til algoritmen ville vært en utjevningsalgoritme som fjerner trappeffekten. Vi tenker oss en metode er å tegne opp en resultant vektor mellom midtpunktene til to rasterceller i randsonen til et område. Det finnes flere metoder for å gjøre slike tilpassninger. En metode som er mye brukt er Bezier-kurven, som er en spline variant.

Bezier-kurven beskrives med et startpunkt og et sluttpunkt, samt mellomliggende hjelpepunkter som beskriver tangentvektorene til endepunktene. Linjesegmenter tegnes som tidligere mellom hvert punkt, men dette er en metode for å kurvetilpasse en randlinje over flere rasterceller som er definert som randceller for et område.

Gitt N + 1 hjelpepunkter kan en Bezier-kurve beskrives matematisk:

$$B(u) = \sum_{k=0}^{N} P_k \frac{N!}{k!(N-k)!} u^k (1-u)^{N-k} \Big\} 0 \le u \le 1$$

Figur 28: Bezier-kurve eksempel N=4



Uansett metode så blir ikke arealet ”sant” ut fra et en definisjon at areal er summen av rasterceller med samme verdi.
Snakker vi egentlig om et sant areal når det gjelder arealet av ”like” nedbørdata?
Nedbørdata er tilnærmet via interpolasjon til en verdi som representerer et areal på 1x1 km.
Spørsmålet burde heller være hvor stor feil vi innfører ved vektorisering.
Generelt vil vektoriseringsalgoritmen trekke ut alle typer geometriske objekter som linjer og punkter som vi kan kalle singulariteter sett i forhold til et areal. Ved å spesifisere kun areal som uttrekksobjekt i en vektorisering går ytterligere data tapt.

Ved klassifisering av nedbørdata antas en reduksjon i tap av nedbørdata pga av singulære geometrier (punkter, linjer). Dette avhenger av hvor grovkornet vi klassifiserer nedbørdata.

Ved en finere inndeling vil der uansett være et potensial for tap av data.
En ytterligere fordel med klassifisering av data er en raskere eksekvering av algoritmen.

Det fines flere tilsvarende algoritmer med deres styrker og svakheter, som for eksempel ”Moore-Neighbor Tracing”, ” Radial Sweep”, ” Theo Pavlidis' Algorithm” og ”Canny edge detection”.

5.2 Kontur

kontu'r *m1 (gj fr fra it. contornare 'omgi, omringe') omriss øyne k-ene av landskapet i mørket / grunntrekk, hovedlinje ane k-ene av partiets nye politikk*
Se også http://www.dokpro.uio.no/.

Konturkart benyttes ofte synonymt med høydekurvekart, som er et topografisk kart.
I motsetning til flatekart (se forrige avsnitt), så er det ikke krav om at grensene skal være lukket, altså en polygon. Det kan være polygon, linjer, og i simpleste forstand et punkt, som angir et markant skille i data.

En definisjon på topografisk kart:

Et topografisk kart viser formen til en terrengoverflate samt et utvalg av de objektene på terrengoverflaten som er lett synlige.

Høydekurver har sin anvendelse for kart i målestokker større enn 1:250 000. Dersom

målestokken blir mindre, blir ekvidistansen (den vertikale avstanden mellom høydekurvene) så stor at kurvene uttrykker lite om terrengoverflatens topografi.
Dette forhold lar seg lett demonstrere med utgangspunkt i et resonnement om grafisk minste mulige ekvidistanse.

Figur 29: Optimal ekvidistanse

$$e = \frac{0.5 \cdot M \cdot \tan \alpha}{1000} \text{meter}$$

Et problem med bruk av høydekurver er at det er de bratteste terrengområder som blir dimensjonerende for ekvidistansen. I praksis er det ofte slik at det er de minst bratte partier som er av størst økonomisk interesse, og følgelig burde disse områdene også vært med å bestemme ekvidistansen. Det endelige valget av ekvidistanse vil måtte bero på en samlet vurdering av de topografiske forhold i det landområdet som skal kartlegges [Bjørke, 2005].

Ovenstående tekst er en konvensjonell assosiasjon av kontur og kart. Kontur en term man benytter i mange sammenhenger.
Bruk av kontur er for å klassifisere nedbørdata i bestemte intervaller. Konturer kan brukes som grafisk representasjon for mange fenomener og objekter.

I sammenheng med vektorisering er kontur synonymt med grense- eller skillelinjer.
Hva er det vi prøver å markere med konturer?

Således er egentlig problematikken identisk med foregående avsnitt.
Opptegning av konturer og linjesporing er synonyme problemstillinger når vi snakker om vektorisering av rasterdata. Hver rute i et raster har en verdi, og problemet er da å bestemme hvilke verdier hører sammen (klassifisering).
Grensene er bestemt av rutesett som skiller en mønsterkomponent fra en annen. Å finne disse er beskrevet i forrige avsnitt, men et viktig poeng er at disse rutene må være sortert i riktig rekkefølge hvis vi skal ekstrahere et generelt mønster fra rasterdata.

Generelt kan vi si at det å tegne konturer er en av mange preprosesseringsteknikker for å trekke ut generell informasjon om mønsterkomponenter innen rasterdatabehandling.
Når konturen av en mønsterkomponent er gjenkjent, vil karakteristikken (grafisk fremstilling) til mønsteret bli vurdert og klassifisert. Korrekte konturer vil øke sjansene for gjenskape mer korrekte geometriske objekter.
Spørsmålet man kan stille seg er: hvorfor bruke datakraft for å prosessere rasterdata med linjesporingsalgoritmer for å finne konturer? Hvorfor ikke trekke geometriske objekter rett ut fra mønsterkomponenten?
Grenserutene i en mønsterkomponent er en delmengde av ruter for hele mønsterkomponenten, og således vil man ha et kjennskap til mønsterkomponentens indre ved å kjenne det ytre. Datamengden reduseres vesentlig i de fleste tilfeller, og geometriske gjenkjenningsalgoritmer er enklere og raskere når inndata som skal prosesseres er basert på konturdata fra samtlige ruter i et mønster.

En konklusjon er at linjesporing eller konturalgoritmer er et effektivt bidrag for å gjenkjenne geometriske objekter, og helt essensielt i mønstergjenkjenning.

I nedbørkart bruker man ikke ordet kontur, men isolinjer. I nedbørsammenheng er disse isolinjer mer korrekt benevnt isohyet (linjer trukket gjennom steder med samme nedbørmengde).

Ekvidistansen er en type konvensjonell beregning. Ekvidistansen i konvensjonelle topografiske kart er også utgangspunktet for å beregne hvilken vei nedbøren vil renne av mot havet fra et nedslagsfelt. Det er en aktuell problemstilling for mange hydrologer.

5.3 TIN modell – avledet av Delaunay Triangulering(DT).

Hva er TIN?

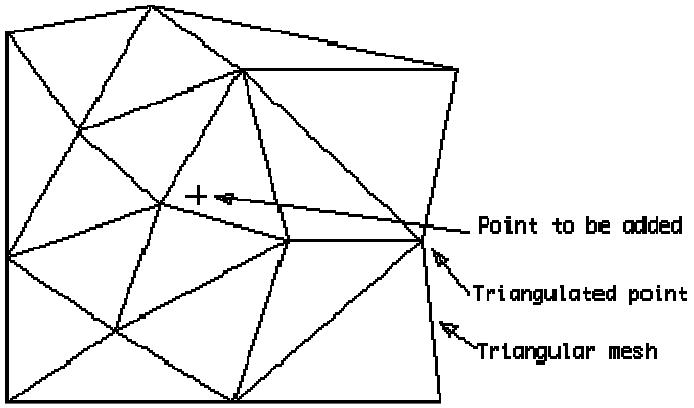
På tilsvarende måte kan vi representere en litt ruglete flate i tredimensjonalt rom - som f.eks. terreng - ved å representere et nennsomt utvalg av punkter og legge trekanter mellom tre og tre nabopunkter. Beliggenheten av punkter på de plane trekantene kan bli funnet ved hjelp av enkle regnestykker ut fra beliggenheten av de tre hjørnepunktene.
Denne representasjonsformen går under Triangulated Irregular Network (TIN) [Albregtsen, 2006].

TIN beskriver terrenget med trekantflater eller terrengfasetter i 2D eller 3D som geometrisk representasjon. TIN modellen er ofte brukt til å produsere digitale terrengmodeller. Fordelen med TIN modellen er en avledet digital datastruktur som er veldig effektiv og krever mindre lagerplass enn en regulær rutenettstruktur.
TIN binder sammen irregulær plasserte punkter i terrenget i trekanter. Punktene utgjør hjørnepunktene i trekantene. Trekantene selv utgjør polygoner hvor sidene er lenkede linjesegmenter hvor hjørnepunktene er de eneste knutepunktene. Hver triangel kan behandles som en flate eller fasett med geometrisk utstrekning entydig definert med kartesiske koordinater (x,y,z). Tettheten av triangler er kun bestemt av tettheten av punktdata, i motsetning til datagrunnlaget, hvor punktdata eksisterer som et sett punkter i et regulært rutenett. Algoritmer for å kalkulere stigning, flatevinkling for å automatisk generering av konturer, perspektivkart, skyggelegging med mer, drar nytte av TIN modellens lagringsstruktur.

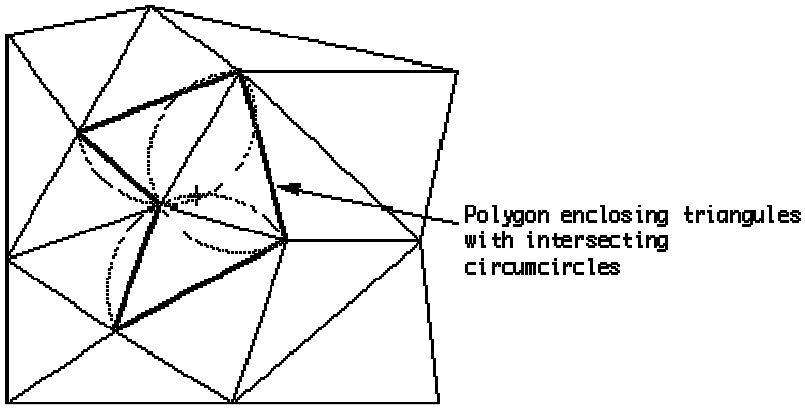
Den mest kjente metoden for utvikling av en TIN modell er DT (Delaunay triangulering). Metoden er kjent som en måte for å maksimere minimumsvinkelen i en trekant. Den utvikler et unikt sett med trekanter som er optimalisert mot å være mest mulig likesidede.

Figur 30, Figur 31 og Figur 32 viser stegvis metode for å bestemme punkter i en trekant [Bourke, 1989]:

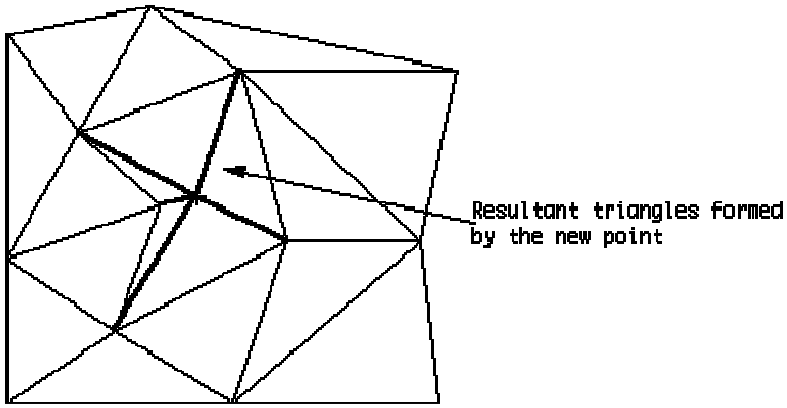
Figur 30



Figur 31



Figur 32



Algoritmen kan beskrives med følgende pseudokode [Bourke, 1989]

```
subroutine triangulate
input : vertex list
output : triangle list
   initialize the triangle list
   determine the supertriangle
   add supertriangle vertices to the end of the vertex list
   add the supertriangle to the triangle list
   for each sample point in the vertex list
      initialize the edge buffer
      for each triangle currently in the triangle list
         calculate the triangle circumcircle center and radius
```

```
         if the point lies in the triangle circumcircle then
            add the three triangle edges to the edge buffer
            remove the triangle from the triangle list
         endif
      endfor
      delete all doubly specified edges from the edge buffer
         this leaves the edges of the enclosing polygon only
      add to the triangle list all triangles formed between the point
         and the edges of the enclosing polygon
   endfor
   remove any triangles from the triangle list that use the supertriangle
vertices
   remove the supertriangle vertices from the vertex list
end
```

Et generelt problem ved Delaunay-triangulering og operasjoner på TIN er at det kan oppstå spesialtilfeller som må takles omhyggelig for å unngå tallmessige problem på grunn av tvetydighet og nulltrekanter [Bjørke, 2005].

Vi kan betrakte nedbør som topografisk fenomen i 2D (trekantflater representerer en enkelt verdi) eller 3D (2.5 D - trekanter som skråflater). I 3D kan ethvert punkt i skråflaten bestemmes for eksempel ved lineær interpolasjon.

For å få en best mulig tilnærming, bør skarpe kanter i den ruglete flaten følges av "skjøten" mellom to trekanter, og jo mer ruglete flaten er i et område, jo mindre bør trekanten være [Albregtsen, 2006].

Interpolasjon i et triangelnett er en fasettmetode. Det innebærer at trianglene får tilordnet hver sin interpolator, og at en interpolator ikke benyttes utenfor det triangel den er tilordnet.

Ved lineær interpolasjon er det mest nærliggende å velge en lineær interpolator gitt ved

$h = a_0x + a_1y + a_2.$

Dette er likningen for et plan hvis parametere bestemmes på grunnlag av vedkommende triangels tre hjørner. Ved å transformere til et lokalt koordinatsystem slik at

$y_1 = x_1 = x_2 = 0$, blir løsningen for parametrene meget enkel. Vi får
$a_2 = h_1$,
$a_1 = (h_2 - h_1)/y_2$ hvor $y_2 \neq 0$,
$a_0 = (h_3 - a_1y_3 - h_1)/x_3$ hvor $x_3 \neq 0$.

Som vi ser, vil likningssystemet ikke gi en entydig løsning dersom de tre punktene er kollineære (punktene ligger på den samme rette linjen) eller om punktene ligger i et vertikalt plan. Selv om det er en ulempe at vertikale plan ikke lar seg definere i en 2.5D terrengmodell, vil dette normalt ikke være noe stort praktisk problem [Bjørke, 2005].

I forkant av TIN modellering må punktene som skal inngå i et TIN hensiktsmessig bestemmes (klassifiseres). Jeg kunne for eksempel basert bestemmelsen av signifikante punkt som i foregående avsnitt. Det gir liten mening å foreta en TIN modellering med alle punktene i rutenettet for nedbør. Prosessen med å velge ut signifikante punkt kalles for tynning.

I [Heywood, 2006] nevnes fire vanlige metoder for tynning. Disse metodene kalles skeleton, filter eller VIP (Very Important Point), hierarchy og drop heuristic. The skeleton method - fritt oversatt til skjelettmetoden - er kanskje den som passer best for problemstillingen med nedbørdata i et regulært rutenett. Metoden innleder med å lage et "vindu" på 3 x 3 ruter. Dette vinduet passeres over hver enkelt rute og sammenlikner den med naborutenes verdi. Et signifikant toppunkt bestemmes ut fra om ruten, som nå befinner seg i sentrum, har høyeste verdi av alle rutene i vinduet. Omvendt – det er et bunnpunkt hvis de andre rutene har høyere verdier. Ikke signifikant punkt er det hvis ingen av de foregående kriteriene er oppfylt. I neste omgang gjentas dette med de signifikante punktene og med et vindu på 2 x 2 ruter. Toppunkter blir så lenket sammen, og det samme gjør bunnpunkter, og en linjetynningsalgoritme bør benyttes for å bestemme de mest kritiske punktene som utgangspunkt for TIN modellering. Tynningsalgoritmen er følsom for støydata og er veldig CPU intensiv.

Mer pragmatiske metoder kan benyttes som utgangspunkt for TIN modellering for å bestemme signifikante punkter, nemlig å klassifisere nedbørdata og for eksempel benytte et konturkart hvor punkter på isolinjer er signifikante punkter.

6 GRASS

Geographic Resources Analysis Support System med akronymet GRASS er et Geografisk Informasjons System (GIS) for håndtering av geospatiale data. Håndtering av geospatiale data inkluderer blant annet metoder for bildebehandling, grafikk/kart produksjon, romlig modellering, grafisk presentasjon samt import- og eksport av raster- og vektordata i forksjellige formater. GRASS har en intern lagerstruktur for både raster- og vektordata med tilhørende geografisk referanse.

GRASS benyttes i dag av akademiske institusjoner, kommersielle aktører, statelige etater og miljøvernorganisasjoner.

GRASS er åpen kildekode og distribueres under GNU GPL lisens. Det betyr at det er gratis å laste ned å kan brukes fritt. Eventuelle endringer i kildekode må derimot publiseres og distribueres med GRASS systemet for å være godkjent som en del av GRASS.

GRASS er godt dokumentert og der er i tillegg skrevet mange artikler og såkalte "tutorials" som har til hensikt å gi en praktisk innføring i bruk av GRASS.

GRASS består av et hundretalls applikasjoner. Mange applikasjoner er bygd på annen åpen kildekode eller bibliotek, eller er implementert som skallskript som omslutter andre applikasjoner av en generell karakter dvs. applikasjoner som kan benyttes til flere formål. Eksempelvis har vi allerede nevnt GDAL (Geospatial Data Abstraction Library) og OGR (OpenGIS Simple Features Reference) et subbibliotek til GDAL

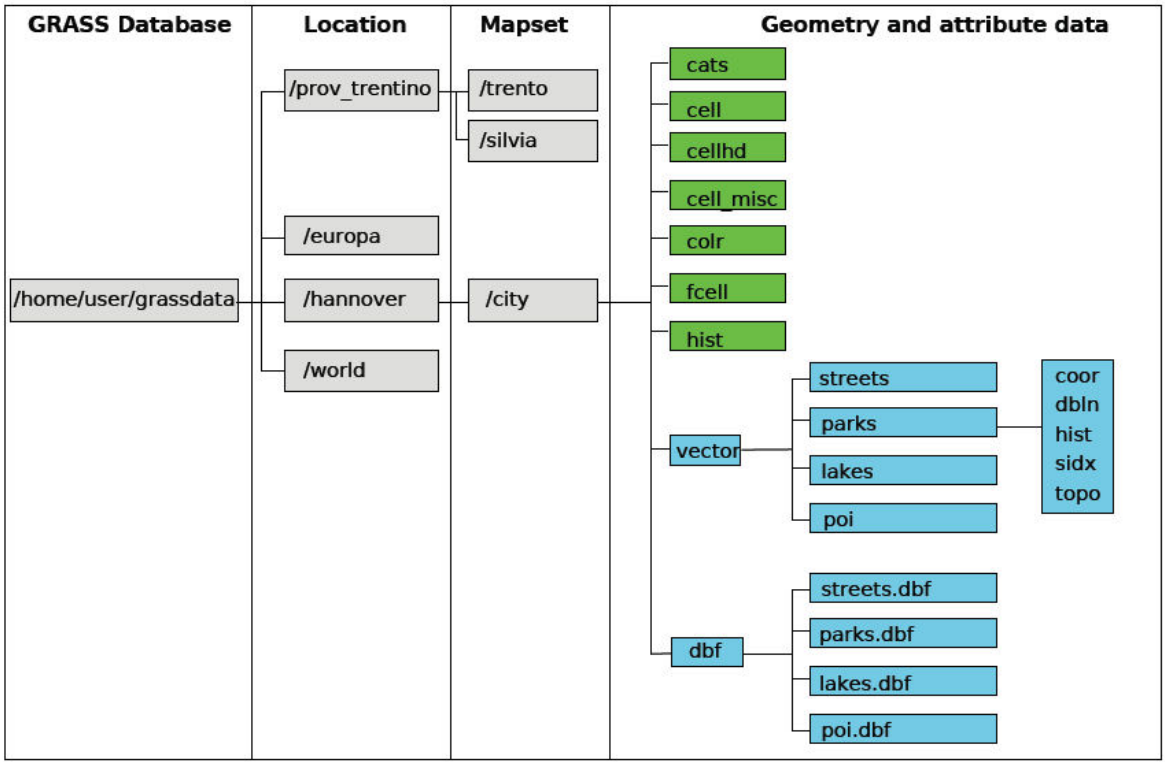
Figur 33 gir en oversikt over organiseringen av kommandoene i GRASS [Dassau, 2005].

Figur 33: GRASS applikasjonslag

Prefix	Functional Group	Description	Example Commands
d.*	display	Graphical display and visual query	d.vect, d.what.rast
db.*	database	Database management	db.select, db.copy
g.*	general	General file management	g.remove, g.proj
i.*	imagery	Image processing	i.rectify, i.fft
ps.*	postscript	Map design for PostScript format	ps.map
r.*	raster	Raster data processing	r.buffer, r.cost
r3.*	voxel vector	3-D raster data processing Vector data processing	r3.mapcalc, r3.out.vtkv.* v.net, v.digit

GRASS installeres med innbygget DBMS (Database Management System) basert på dBase hvor geospatiale attributtabeller er lagret i DBF (DataBase File) filer. GRASS har begrenset SQL (Structured Query Language) støtte mot sin innebygde DBMS. Imidlertid har GRASS drivere mot eksterne DBMS, som PostgreSQL/PostGIS og MySQL for å nevne et par.

Figur 34: GRASS filstruktur [Dassau, 2005]



GRASS har både et grafisk- og tekstbasert brukergrensesnitt som startes med hhv. grass –gui og grass –text.

GRASS systemet er i konstant utvikling, og jeg vil for orden skyld påpeke at jeg i skrivende stund referer til 6.0.2 versjonen her og i påfølgende avsnitt.

6.1 Bruk av GRASS for bearbeiding av nedbørdata

GRASS må kunne kalles et relativt komplett GIS verktøy. Arbeidet med nedbørdata i GRASS har bestått i:

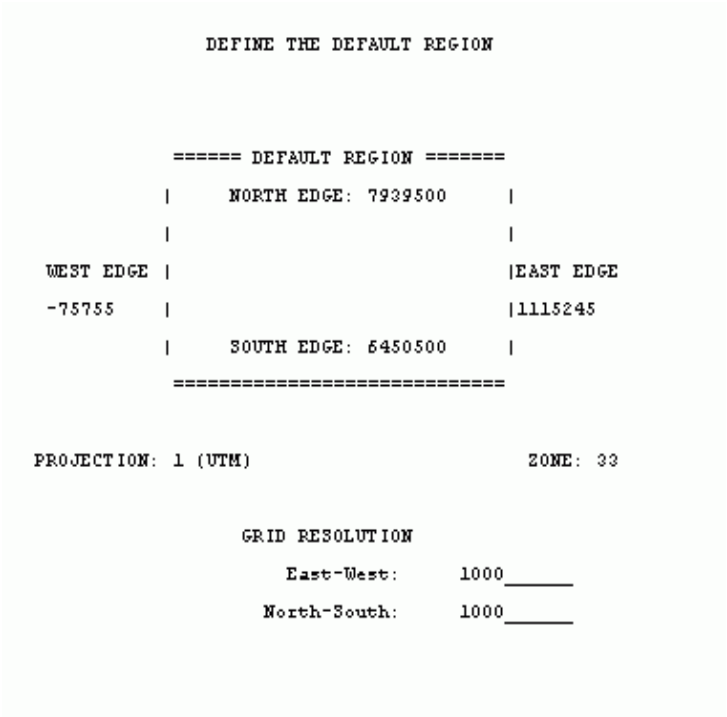
- datautvelgelse – import av rasterdata i arbeidsområdet.
- transformasjon og editering – interaktiv editering av data, konvertering mellom raster- og vektordata og terrengmodellering på vektorformat for som underlagsdata for forskjellige representasjoner av nedbør.
- analyse ved hensiktsmessig klassifisering av nedbørdata. Eksport av data på forskjellig vektorformat for lagring i PostGIS og applikasjoner.
- presentasjon – visualisere stadier i arbeidet for å gi en ide om hvordan visualisering i nettleser vil bli, og for å verifisere operasjoner i GRASS. Eksportering av data på vektorformat for visualisering i nettleser.
- SQL grensesnitt – søk etter data basert på egenskaper og geografisk informasjon.
- automatikk – utvikle rutiner for som automatiserer flere rutinene ovenfor.

Før vi starter å bearbeide nedbørdata må man etablere et arbeidsområde.
Et arbeidsområde i GRASS betyr at man må definere en database, lokasjon og kartdatasett.
Jeg forutsetter i denne diskusjonen at det er første gang jeg starter opp GRASS.

Termer som beskriver arbeidsområdet i GRASS:

- Database – definere hvor filene for GIS prosjektet for en gitt lokasjon/område samt tilhørende kartdata vil bli lagret. Database må peke til en eksisterende filkatalog før lokasjon og kartdatasett kan defineres.
- Kartdatasett – en samling data innenfor en gitt lokasjon/området. Organisere data av samme type og kategori.
- Lokasjon – definerer projeksjon, romlig omfang av rutenettet og antall punktdata i et rutenett (grid).

Figur 35: geografisk rammeverk for nedbørdata i GRASS



GRASS strukturen i mitt arbeid er definert med LOCATION = rrmnd og MAPSET=2005. Valg av struktur vil bli drøftet etter hvert.

Videre diskusjon baserer seg på interaktiv bruk av GRASS i tekstmodus for å vise applikasjoner hvilke applikasjoner som blir tatt i bruk.
GRASS i tekstmodus er et skallmiljø hvor man med riktig oppsett av GRASS miljøvariable peker til GRASS databasen, lokasjoner, kartdatasett og applikasjonene.
I tillegg er det full tilgang til vanlige UNIX, Linux eller Windows kommandoer avhengig av hvilken plattform GRASS er installert på.

6.2 Import av nedbørdata på rasterformat

GRASS støtter import av rasterdata av mange forskjellige formater. Rasterformater kan deles inn i 3 typer av rasterformater.
Disse er:

- Bildeformat – individuelle raster har alltid positive heltallsverdier i kjente pikselbaserte bilde formater som f. eks PPM (Portable Pixel Map – UNIX), PNG (Portable Network Graphic), JPEG (Joint Photographic Experts Group) og GIF (Graphics Interchange Format).

- ASCII format – individuelle raster kan bestå av enten positive og negative heltallsverdier eller flytall. ASCII-GRID og Arcinfo er eksempler på dette formatet.

- Binærformat – i binær rasterformat kan de individuelle piksler med enten positive og negative heltallsverdier eller flytall også bli lagret i forskjellige kanaler med

forskjellig oppløsning. (Geo)TIFF (Tagged Image File Format) og ERDAS/IMG er eksempler på slike formater.

I GRASS blir rasterformater vanligvis importert med deres iboende geografiske referanseinformasjon (rutenettets geografiske utstrekning) og oppløsning. Se kapittel 2 for beskrivelse av rastermodellen. I GRASS kan man velge å endre formatet på rasterdata til å representere flyttall eller heltall.

I de tilfeller man ønsker å importere rasterdata som ikke inneholder geografisk referanseinformasjon må to hensyn tas i betraktning:

- hvis lokasjon og oppløsning er definert, så vil oppløsningen til det importerte rasterformatet bli konvertert, slik at det samsvarer med den definerte oppløsningen for den opprettede lokasjonen.
- hvis parametrene for lokasjonen samsvarer med kartet som skal importeres, spesielt oppløsningen, så må kartet importeres uten å gjøre forandringer på importparametrene.

Dette er ikke tilfellet her. Nedbørdata inneholder nødvendig geografisk informasjon og en definert oppløsning på rutene i rutenettet. Se forrige kapittel om bruk av `gdalinfo`.

GRASS støtter import av flere forskjellige rasterformater via GDAL biblioteket. Se Figur 36 for alternative formater i gjeldende versjon av GRASS.

Figur 36: Raster importapplikasjoner [Dassau, 2005]

GRASS Module command	*Import Raster format*
r.in.ascii	GRASS ASCII
r.in.bin	BIL, GMT binary files, LANDSAT TM5
r.in.gdal	ARC/INFO ASCII/Binary GRID, BIL, ERDAS (LAN, IMG), USGS DOQ, JPEG, SAR CEOS, EOSAT, GeoTIFF, PPM/PNM, SDTS DEM, GIF, PNG (see also `http://www.gdal.org/formats_list.html`)

Nedbørdata er i henhold til figur definert som ASCII/Binary GRID format. Import av nedbørdata i GRASS utføres med `r.in.gdal`. Her er måten jeg har benyttet applikasjonen på.

```
r.in.gdal -oe input=rr/rr_jan_2005 output=1
```

Nedbørdata på GRASS rasterformat lagres i GRASS databasen i lokasjon ”rrmnd” og kartsett ”2005”.

Valg av tall som navn på kartsett og rasterdata er gjort med henblikk på senere automatisering av rutiner, og av temporale hensyn noe som vil drøftet senere i dette kapitlet.
For å verifisere import av rasterdata benyttes den generelle kommandoen `g.list` .

```
g.list rast
----------------------------------------------
```

```
raster files available in mapset rrmnd:
1

----------------------------------------------
```

Jeg gjør oppmerksom på at det er rasternavnet som er listet, og ikke antall!

Med kommandoen `g.region` kan jeg sjekke geografisk informasjon samt rasterinformasjon for lagrede rasterdata.

```
g.region -p rast=1

projection: 1 (UTM)
zone:       33
datum:      wgs84
ellipsoid:  a=6378137 es=0.00669438
north:      8000500
south:      6449500
west:       -75755
east:       1140245
nsres:      1000
ewres:      1000
rows:       1551
cols:       1216
```

Her følger alternativ utlisting av avgrensningsboks (eng. bounding box) for nedbørdata. Forskjellen her er blant annet at vi har omformet UTM-koordinater til lengde- og breddegrad.

```
g.region -pl rast=1

long: -1.35045 lat: 71.40000 (north/west corner)
long: 33.07354 lat: 71.23790 (north/east corner)
long: 25.77755 lat: 57.72743 (south/east corner)
long: 5.28928 lat: 57.81458 (south/west corner)
rows:       1551
cols:       1216
Center longitude: 15:41:50.926211E [15.69748]
Center latitude:  64:32:41.922283N [64.54498]
```

6.3 Grafisk presentasjon i GRASS

GIS dreier seg mye om grafisk presentasjon, og for hver operasjon som avleder et nytt kart, ønsker man en mulighet til å se resultatet presentert grafisk. Grafisk presentasjon av kart betyr å gjøre en del valg. En ting er hva som skal visualiseres basert på geografisk informasjon, men like viktig er fargevalg. Mennesker har alltid en persepsjon om farger benyttet til ulike formål. For nedbørdata er det kanskje rimelig å assosiere vann, nedbør som en variasjon i blått. De dypeste havdyp som mørk blått er noe som fester seg i minnet etter mange geografitimer i skolen med å studere kart.
En like viktig ting er håndtering av såkalte NULL data. NULL data er raster uten verdi ut fra en definert kontekst. Fastlandet er vårt er ikke rektangulært, allikevel er rasterdata formatert som en rastermatrise på 1216 x 1551 punkter. Mange av disse punktdataene er NULL data i en kontekst der data representerer interpolerte nedbørdata over fastlandet. GRASS er fleksibelt og har muligheter for å endre på NULL data konteksten – f. eks inkludere ”vestlige” NULL data som hav – hva da med NULL data mot øst?

Poenget her er at enten ignorerer man NULL data i rasterdata, eller du håndterer dette eksplisitt og gir det en kontekst.

For å visualisere rasterdata og generelt kunne visualisere resultatet av enhver operasjon man foretar seg med data starter man GRASS skjermhåndterer `d.m` :
Det grafiske grensesnittet i GRASS støtter bruk av vinduer i X11 og Windows gjennom drivere for de respektive plattformene. I tillegg finnes grafisk driver (PNG) for å lagre grafikken på en .png fil for å inkludere i rapporter, artikler eller i en masteroppgave.

Kjør d.mon –help for å lese mer om monistorering i GRASS.

Kommandoen d.mon PNG starter driveren for å skrive til en .png fil.

For PNG driveren er det definert tilhørende miljøvariable. For umiddelbar lagring av enhver operasjon settes miljøvariabelen GRASS_PNG_AUTO_WRITE=TRUE, ellers skrives ikke standard filen map.png før man stopper driveren med `d.mon stop=PNG`.

Skjermhåndtereren holder rede på hvilke drivere som er startet og hvilken som er aktiv eller gjeldende driver for å motta neste grafiske operasjon. For å visualisere rasterdata på skjermen i et vindusmiljø, bruker man kommandoen `d.mon x0`. Intuitivt skjønner man at der er mulig å starte flere skjermvinduer, og det er riktig.

For å starte grafiske operasjoner i GRASS bruker man hhv `d.rast` for rasterdata og `d.vect` for vektordata. I tillegg er der en grafisk operasjon kalt `d.frame` som evner å splitte opp et grafisk vindu i rammer for å separere grafisk informasjon i et ønsket/valgt utlegg.

For rasterdata brukes kommandoen `r.colors` for å definere fargene i et rasterkart. Kommandoen endrer fargetabellen til rasterkartet og er således en permanent endring til eksplisitt endret på nytt med samme kommando. Der er flere fargeapplikasjoner i GRASS som benytter forskjellige datatyper som inndata og har forskjellig intensjon med bruk av farger. Leserens oppfordres til selv å studere GRASS manualen [se http://grass.itc.it/grass60/manuals/html60_user/index.html] om bruk av farger.
Modulen `r.colors` har en rekke predefinerte konvensjonelle fargevalg, f. eks et fargevalg for høyder i topografiske kart, men aksepterer også brukerspesifiserte fargevalg som inndata.

Å legge til fargevalg for vektordata gjøres ved endring av attributtabellen for et vektorkart. Typisk vil attributtabellen for et vektorkart være bygd opp av minimum to kolonner, nemlig en kategoriattributtkolonne og en attributtkolonne for måleverdi. I GRASS legger man inn fargevalg ved å utvide attributtabellen med et attributt kalt GRASSRGB som er RGB kodet. RGB er en additiv fargemodell hvor fargen er en sum av primærfargene rød, grønn og blå. Formatet på attributtsverdien i denne kolonnen er 'RRR:GGG:BBB'[5].

5 En erfaring jeg gjorde er at GRASSRGB attributtet kan skrives på formen 'RRR, GGG, BBB' i stedet for, og som det står i manualen 'RRR:GGG:BBB'.
Dette er mer i overensstemmelse med SVG blant annet og gjør at man kan redusere prosesseringstid som går med til formatering.

Når det gjelder fargevalg i GRASS så er det å si at der er ingen automatikk i overføring av fargevalg ved konvertering av raster- og vektordata fra det ene formatet til det andre, samt endringer internt innenfor et format.

NULL data i rasterformat er en predefinert uendelig stor negativ tallverdi (heltall, flyttall). Som regel brukes asterisk-tegnet (’*’) som symbol på NULL data. Der er en rekke kommandoer for å indikere om der er NULL data i datasettet. Applikasjonen `r.report` er en måte å liste ut rasterdatastatistikk på. Applikasjonen `r.null` benyttes hvis du ønsker å redefinere NULL eller sette inn NULL inn i en ny kontekst, ved å gi det en verdi. NULL er også viktig i en annen sammenheng, nemlig til maskering av rasterdata. I GRASS kan man lage en maske ved hjelp av NULL verdier. MASK er et predefinert navn på rasterdata brukt til maskering i GRASS. Når MASK rasterdata er opprettet maskeres implisitt alle grafiske operasjoner. Maskering fjernes ved å slette MASK rasterdatasettet.

Her er et eksempel på å lage en maske:

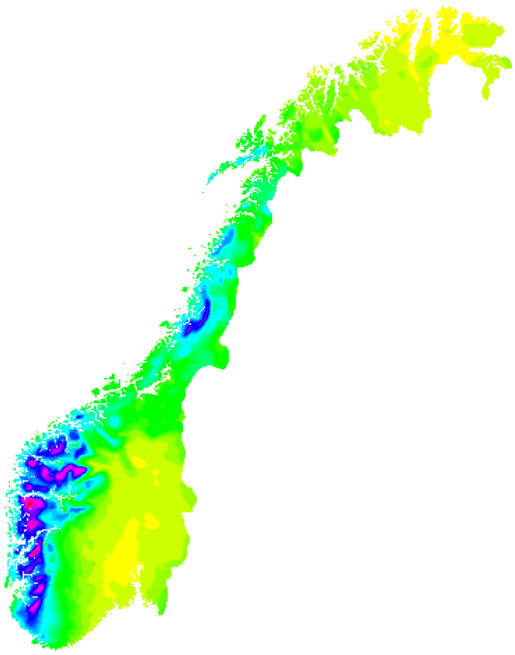
```
r.mapcalc 'MASK=(isnull("1"), null(), 0)'
```

I GRASS vil man ved grafiske operasjoner bli informert om en maske er tilstedet. For å fjerne masken benytter man en generell kommando:

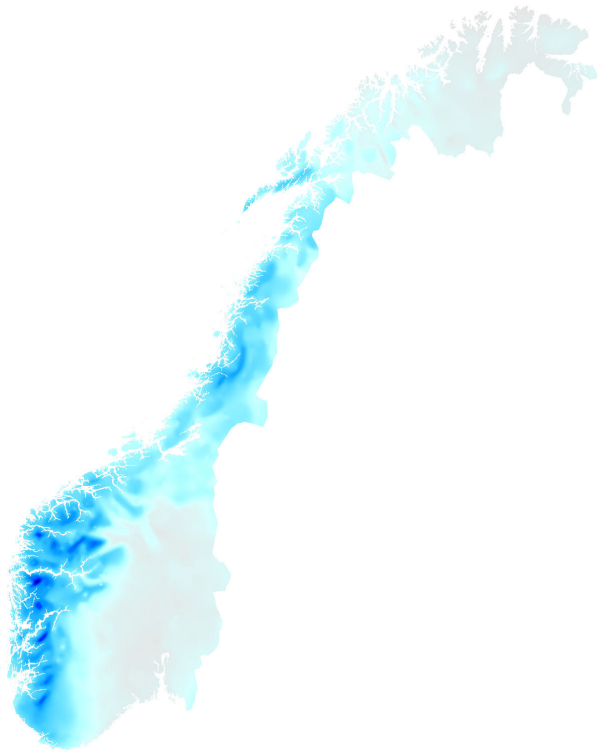
```
g.remove MASK
```

Noen illustrasjoner med rasterdata etter import av nedbørdata i GRASS (se, Figur 37, Figur 38 og Figur 39).

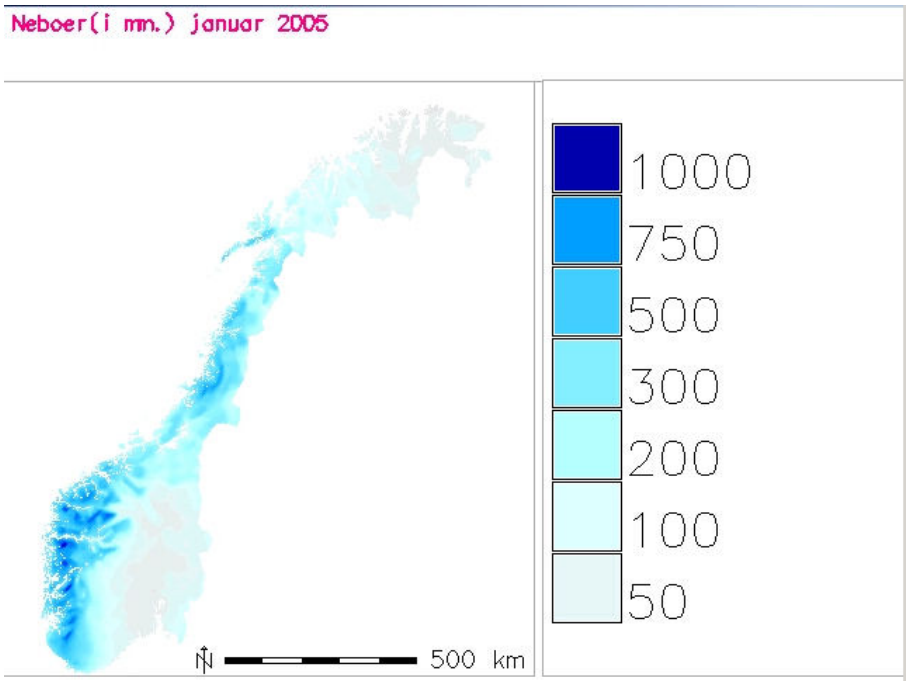
Figur 37: rasterdata rr januar 2005, automatisk fargevalg



Figur 38: Kundespesifiserte farger med r.colors



Figur 39: Eksempel på bruk av d.frame



6.4 Geometrisk 2D modellering i GRASS

I kapittel 5 drøftet jeg vektorisering og geometrisk modellering. Hensikten med modellering er lage en grafisk representasjon av nedbørdata.

Se igjen Figur 1 for hvordan Klimadivisjonen visualiserer nedbør. Dette er som tidligere nevnt et koropletkart eller et fargelagt flatekart.
I dette avsnittet skal vi se på hvordan gjøre dette i GRASS, og gi visuelle eksempler på nedbørdatamodeller.

En GIS løsning må supportere lagring av kartdata på både raster- og vektorformat med henblikk på geometrisk modellering. Kompleksiteten i algoritmer for geometrisk modellering er bestemt av kartformatet. Visse algoritmer utføres raskere når formater er raster, mens andre algoritmer er lettere å implementere og kjører bedre når kartdata er på vektorformat.

Jeg har importert et år med månedsnedbør som rasterdata inn i GRASS. GRASS kan altså importere griddede nedbørdata.

I påfølgende avsnitt har jeg lagret vektordata på forskjellig format for å illustrere hvordan man kan konvertere rasterdata til vektordata samt å konvertere en vektortype til et annet.

Før jeg starter med å drøfte vektorisering av rasterdata, så er det hensiktsmessig å bestemme hvilke typer kart som ønskes laget.

Å klassifisere nedbørdata i intervaller både er hensiktsmessig og en konvensjonell metode. Dette reduserer mengden av vektordata som må lagres.
Prosessen i GRASS kan sorteres i følgende kulepunkter:

- Formatering av nedbørdata fra flytall til heltall – vi er ikke interessert i desimaler. Enklere format og informasjon før reklassifisering
- Reklassifisering av nedbørdata – grovere inndeling av nedbørdata i intervaller.
- Vektorisering og generalisering av de grovinndelte nedbørdata til geografiske verdier som areal og linjer (punktdata er uinteressant i denne sammenhengen).
- Fargekoding av nedbørdata på vektorformat.

Eksemplene og teksten som følger er en drøftelse basert målsettingen om å lage et tradisjonelt koropletkart basert på nedbørdata.

Vi bør kjenne til innholdet i datagrunnlaget på rasterformat. Informasjon om lagrede rasterdata(se `r.info`).vises i Figur 40.

Det som i første omgang er interessant her, er maksimum og minimum av nedbørverdier. Reklassifisering av nedbørdata utføres for å redusere datamengden uten å ødelegge informasjon utover det som kan forsvares.
Rasterdata er som lagret som flyttall, og vi benytter r.mapcalc for å lage et nytt datasett på rasterformat med heltall. Dette gjør i GRASS ved hjelp av kartalgebra:

```
r.mapcalc 'r_jan200_int)=int(r_jan2005)'
```

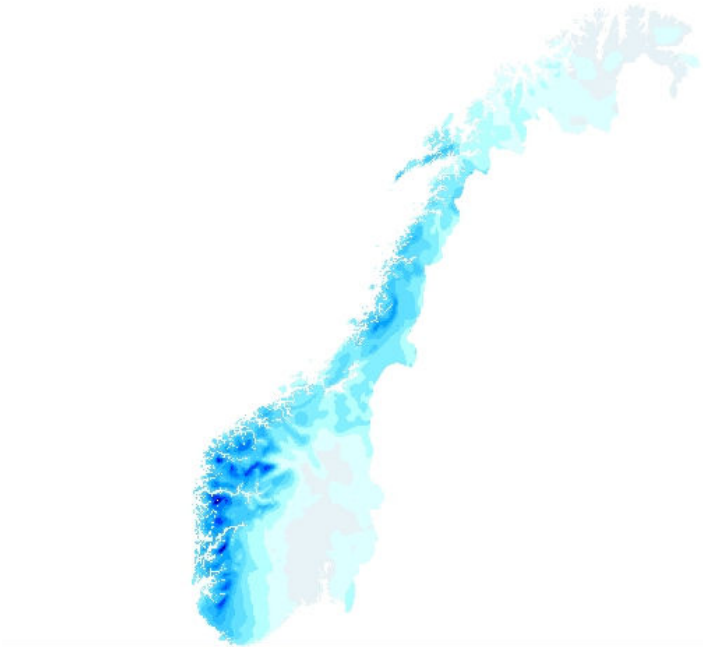
Figur 40: r.info r_jan2005

```
+----------------------------------------------------------------------------+
| Layer:    r_jan2005                       Date: Tue Aug 29 10:45:29 2006   |
| Mapset:   rr                              Login of Creator: leifgo         |
| Location: norge_wgs84                                                      |
| DataBase: /usr/local/share/grass                                           |
| Title:     ( r_jan2005 )                                                   |
|----------------------------------------------------------------------------|
|                                                                            |
|   Type of Map:  raster              Number of Categories: 255              |
|   Data Type:    FCELL                                                      |
|   Rows:         1551                                                       |
|   Columns:      1216                                                       |
|   Total Cells:  1886016                                                    |
|        Projection: UTM (zone 33)                                           |
|             N:    8000500    S:    6449500   Res:  1000                   |
|             E:    1140245    W:     -75755   Res:  1000                   |
|   Range of data:    min =  2.943588 max = 1007.231323                      |
|                                                                            |
|   Data Source:                                                             |
|                                                                            |
|                                                                            |
|                                                                            |
|   Data Description:                                                        |
|    generated by r.in.gdal                                                  |
|                                                                            |
|                                                                            |
+----------------------------------------------------------------------------+
```

Nedbørdata blir reklassifisert i ulike intervaller – finere inndeling med lave nedbørmengder og større intervaller ved større nedbørmengder (fra 100 mm og oppover).

Klassifisering av nedbørdata i intervallet 0 – 100 mm i steg av 50 mm og videre i steg av 100 mm intervaller utfører vi i GRASS med hjelp av modulen `r.recode`. Det øverste intervallet er fra 1000–1100 mm.

Figur 41: 1_50_1100



Rasterdata lagret i 1_50_1100 er nå utgangspunktet for vektorisering.

En liten digresjon om kartalgebra (en sentral og nyttig applikasjon i GRASS):

Kartalgebra-applikasjonen i GRASS er utført med operander på rasterformat.
Før klassifisering kunne vi utført en rekke relevante algebraiske operasjoner med `r.mapcalc` for å lage et datagrunnlag før vektorisering.
Årlig statistikk av nedbørdata kan avledes for eksempel årlig midlere nedbør, som en aritmetisk middelverdi av summen av månedsdata (heltallsverdier) som:

```
r.mapcalc 'AM = (int("1") + int("2") + …..int("12"))/12'
```

På samme måte kunne man sette opp et uttrykk for å beregne varians og standardavvik.

Nedbørdata i kombinasjon med temperatur bestemmer klimasoner eller klimatyper.
Den meste kjente og vanlige metoden for å dele verden inn i klimasoner er Köppens klimaklassifikasjon. Sonene er delt inn etter temperatur- og nedbørsnormaler, i samsvar med det som observeres av vegetasjonssoner. K. Wladimir Köppen (ty. 1846–1940) publiserte systemet i den detaljerte formen første gang i 1918 og reviderte det senere flere ganger helt frem mot sin død i 1940.

Klimasoner deles inn i 5 hovedgrupper. Soner som er representative for Norge er C, D og E. Det meste av landet sorterer inn under D, mange kyststrøk under C og Svalbard, deler av Øst-Finnmark samt fjellområder under E (se http://www.met.no/met/met_lex/g_k/Koeppen.html).

1.	A	Tropisk klima
2.	B	Tørt klima
3.	C	Temperert klima
4.	D	Polarklima
5.	E	Arktisk klima

- A Tropisk klima: Middeltemperatur > 18 °C i alle årets måneder.
- B Tørt klima: Et område som har temperatur- og nedbørsforhold i henhold til en av de 3 kriteriene under: [R er årsnedbør i cm og T er årsmiddeltemperatur i °C]:
  R < 2T hvis minst 70 % av nedbøren kommer i vinterhalvåret ·
  R < 2T + 14 hvis nedbøren er jevnt fordelt mellom sommer og vinter
  R < 2T + 28 hvis minst 70 % av R kommer i sommerhalvåret
- C Temperert klima: Temperaturen i årets kaldeste måned er mellom +18 og -3 °C. Nedbørsmengden ligger over grensene for tørt klima (B).
- D Polarklima: Temperaturen i årets kaldeste måned er under -3 °C, i den varmeste over +10 °C
- E Arktisk klima: Middeltemperatur under +10 °C i årets varmeste måned

Kulepunktene kan formuleres algebraisk og beregnes med `r.mapcalc` for å lage klimasonetypekart.

Tidligere i avsnittet drøftet jeg at klassifisering er hensiktsmessig når formålet er å avlede nedbørdata som f. eks flatekart, dvs. nedbørdata representert som georefererte geometriske verdier lik areal/polygon.
Anvendbar problematikk er å definere nedbør innenfor definerte vannskiller, også kalt nedslagsfelt for nedbør. Et område som drenerer til et felles utløpspunkt for sitt avløp, ligger i samme nedbørfelt. Et "helt" nedbørfelt danner et vannskille i øvre deler og har utløp i havet.

6.4.1 Konvertering fra raster til vektorformat

Vektorisering av rasterdata i GRASS utføres med modulen r.to.vect. Vektorisering av raster i GRASS betyr å konvertere rasterdata til geometriske verdier som punkt, linje, polygon. GRASS har egne termer for geometriske verdier, som er de basale geometriene med tillegg av egne geometrier som er kombinasjoner satt sammen av de basale geometrier. Areal er en geometrisk verdi i GRASS, og definert som en sammensetning av grenselinjene for et polygon og et punkt, som er sentroiden til polygonen.

Hensikten med vektorisering er:

- redusere datamengden – geometrisk generalisering
- konvertere nedbørdata til andre vektorformat

Vektorisering i GRASS blir til en diskusjon om modellering fordi vektorisering er utført med et mål for øye, nemlig å lagre nedbørdata på vektorformat som geometriske objekter, for videre modellering, konvertering og eksport. De neste avsnittene vil ta for seg vektorisering og modellering.

Det finnes et utall algoritmer for å konvertering av rasterdata til geometriske objekter på vektorformat. Flatemodeller i GRASS brukes en linjesporingsalgoritme som finner ytterkantene av områder med ”felles” rasterverdi. Enkelt fortalt går det ut på å gjenkjenne ytterkantene på raster som omslutter raster med ”samme” verdi. Se kapittel 5.

Vi skal se på et eksempel for å avlede et flatekart i 2D på vektorformat.

Som raster datagrunnlag for vektorisering brukes de klassifiserte rasterdata.
Rasterkartet som jeg har kalt *1_50_1100* skal konverteres til vektorformat. Applikasjonen `r.to.vect` utfører vektoriseringen.

```
r.to.vect in=1 _50_1100 out=1_50_1100 feature=area
```

For visualisering av vektordata bruker man som tidligere nevnt `d.vect`.

Nedbørdata på vektormat er nå lagret internt i GRASS. Geografisk referanseinformasjon er skilt fra attributter i GRASS lagringstruktur. Knutepunktenes tallverdi i de geometriske objektene er lagret på standard databaseformat (dBase) i DBF tabellfiler. Ekstern database kan også benyttes for å lagre attributtene (for eksempel i PostgreSQL).
Hvis ingenting er spesifisert, så vil GRASS automatisk fargelegge kartet. Dette er forklart tidligere. Jeg vil her kort vise SQL grensesnittet for å endre attributtabeller.

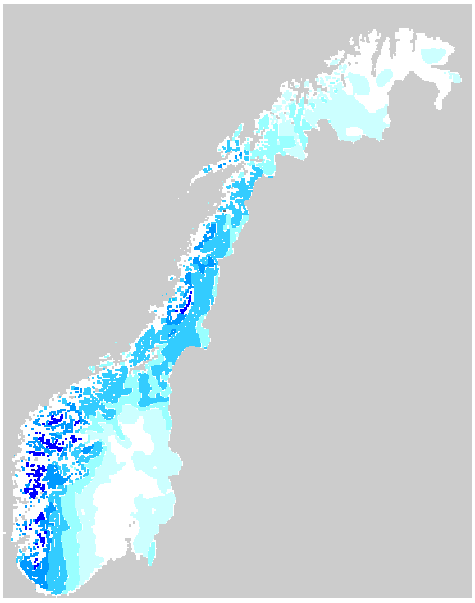
```
echo "ALTER TABLE 1_50_1100 ADD COLUMN GRASSRGB varchar(11)" | db.execute
```

Fargelegging av nedbør er ofte utført med en fargegradient av blåfarge. Fargegraderingen er definert fra hvitt mot mørk blå som representere områder med 0 eller veldig lite nedbør til mer nedbørintense regioner.
Jeg har brukt samme fargepalette som klimadivisjonen bruker. Innsetting av farger utføres med med ”piping” av SQL datautvelgelse til særskilte databaseapplikasjoner i GRASS.

```
echo "update v_jan2005_50_1100 set GRASSRGB='0:153:255' where value >=700"
| db.execute
.
```

… gjentas helt til man har fylt inn samtlige farger for forskjellige nedbørverdier.

Figur 42: v_jan2005_50_1100_polygon



6.4.2 Kontur

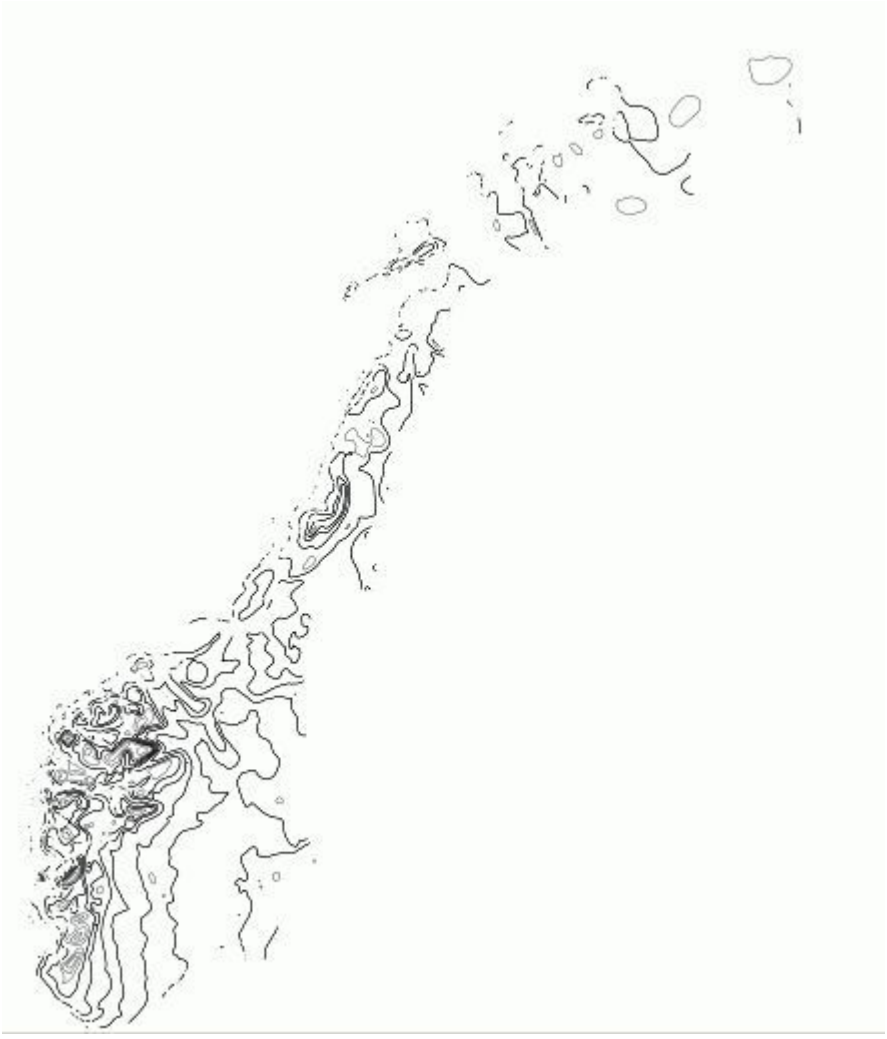
Applikasjonen `r.contour` lager konturkart på vektorformat fra kartdata på rasterformat. Dette er en algoritme som typisk har mindre kompleksitet som følge av rasterformatets regulære lagringsstruktur.
Ved siden av inn- og utdata som standardparametere på kommandolinjen, må man definere ønsket isolinjer (”høydekurver”) og ekvidistansen som ønskes brukt. Ekvidistansen og isolinjer defineres automatisk eller eksplisitt ved å definere intervall(er) som en opsjonell parameter til applikasjonen.

Her har jeg valgt å bearbeide de initiale rasterdata i stedet for de klassifiserte rasterdata som lagret i rasterdata ”1_50_1100”. Begrunnelsen for det er at maksimumsverdien som ligger i området $1000 \leq rr < 1100$ satte høyeste konturverdi til 1100 mm ved forrige klassifisering. Dette kan synes å være en bagatell, men det er feil.

Konturlinjene som markerer maksimumsnivået bør gå igjennom 1000 mm. Dette avledes riktig ved å benytte applikasjonen på de initiale rasterverdiene (se Figur 43).

Det kan bemerkes til Figur 43 at figuren burde ha vært visualisert på et underlagskart med eller uten topografisk informasjon. Det interessante er å se regionale forskjeller i nedbør, og hvor de største nedslagsfeltene for nedbør er, samt å se nedbør i forhold til topografien. Vi ser at det er på Vestlandet vi finner de høyeste største nedbørmengdene, angitt med stor tetthet av isolinjer (isohyet).

Figur 43: v_jan2005_50_1000_contour



6.4.3 TIN (Delaunay Triangulering - DT).

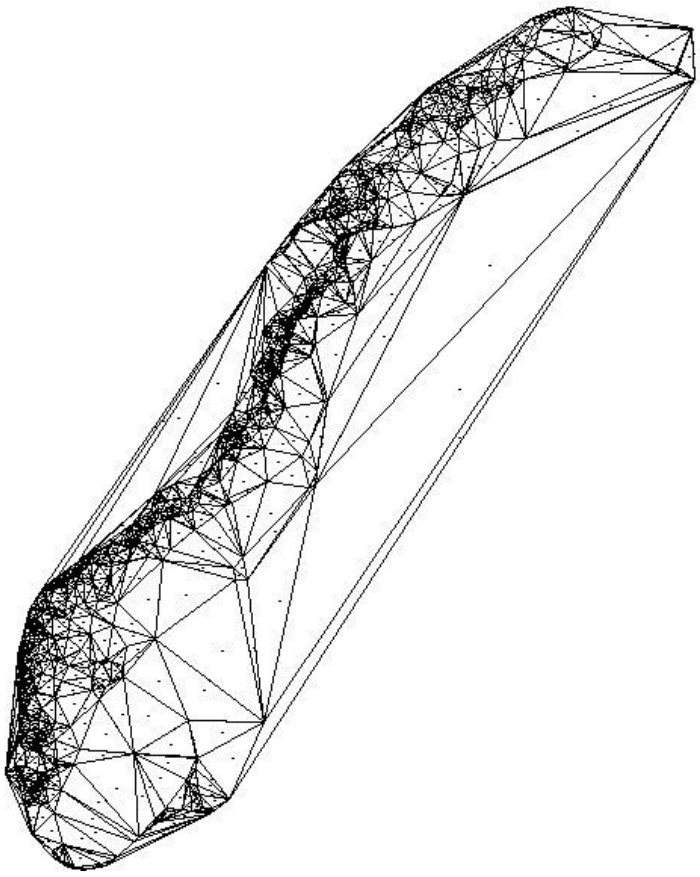
Applikasjon i GRASS som avleder et DT kart kalles `v.delaunay`.
I samme åndedrag nevnes applikasjonen for å lage overleggskart, nemlig `v.overlay`.

Begrunnelsen for bruke begge applikasjonene i denne konteksten er:

1. å få en presentasjon av DT som samsvarer mest mulig med kartet.
2. å demonstrere hvordan man kan lage nye kart ved å legge over hverandre eksisterende kart.

TIN algoritmen tar inndata på vektorformat. På samme måte som tidligere så er det hensiktsmessig å klassifisere data.
Hvis vi hadde benyttet vektorpunkter, så ville vi selvfølgelig fått like mange punkter som nedbørdata på rasterdata. Vi ville fått avledet veldig mange triangler!
Er det en ulempe?
Svaret er både ja og nei – bruk av DT må sees i sammenheng med kartets målestokk og valgt oppløsning. Det er en generell problemstilling å velge oppløsning - se siste avsnitt for en mer fyldig diskusjon når jeg drøfter problemstillinger rundt visualisering.

Figur 44: v_jan2005_50_1100_dt



Figur 44 viser DT av `v_jan2005_50_1100`. Vi gjenkjenner det konvekse ”skroget” som inneholder samtlige punkter som er en del av kartdata.

Matematisk korrekt er å si - det minste konvekse ”skroget” som rommer samtlige punkter i kartdata. Det betyr at det er tillatt at punkter kan ligge på linjesegmentene til det konvekse ”skroget”.

6.4.4 Digital Elevation Model (DEM) av nedbørdata

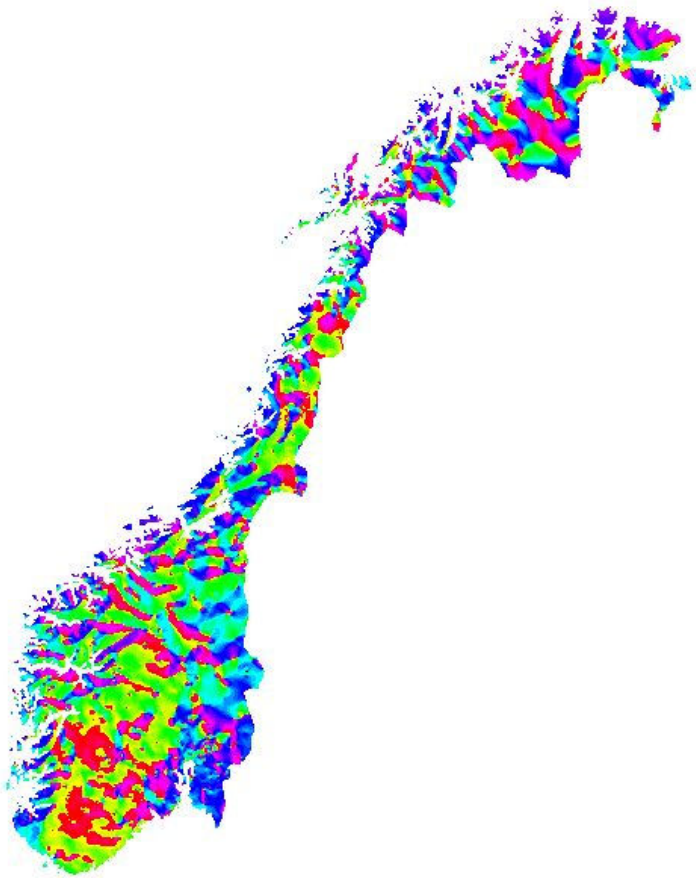
DEM er rasterdata som inneholder høydeverdier. Tradisjonelt tenker man på synlige høyder i terrenget, men det burde være like intuitivt å forestille seg et ”nedbørhøydeterreng”.

De tre neste kartobjektene er laget for å gi en 3D effekt (dypdevirkning) med hhv. `r.slope.aspect` for å lage et kart som beskriver nedbørstigningskart med forskjelllig perspektiv. Stigning (eng. slope) - angis i prosent eller grader (standard: grader), mens aspektkartet (helningsvinkelkart) angir stigningen i grader med en persepsjon av en fast plassert lyskilde (solen) i retning mot klokka med 0 grader i øst.

Vi har hittil sett på geometrisk modellering av nedbørdata i 2D. 2D formatet er gitt med x og y koordinater i projeksjonsplanet og nedbørmengde som et attributt. Slik er lagringsstrukturen i GRASS for nedbørdata på vektorformat.
I 3D eller mer korrekt 2.5D (”sann” 3D innbærer muligheten for flere z verdier over et x,y punkt) må nedbørmengden gjøre om til z koordinat.

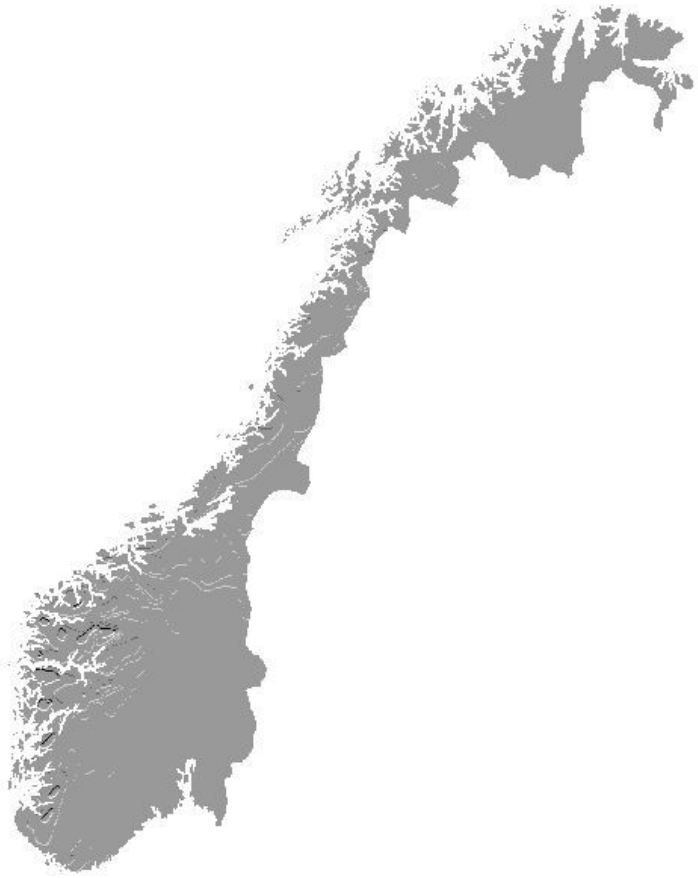
Figur 45: r_jan2005_50_1100_25D_slope

Figur 46: r_jan2005_50_1100_25D_aspect



Det siste eksemplet er et relieffkart – som fremkommer med skyggelegging av le siden av høyder sett fra en lyskilde. Skyggeleggingskart er et godt norsk ord som også brukes. Lyskilden er selvfølgelig vår egen sol hvis posisjon beskrives med asimutvinkel (her sett med 0 i nord, og stigende verdier med klokka) og solhøyden til horisontalplanet. I tillegg har man en z overdrivelsesfaktor man gi inn for å "kaste lengre skygger", dvs. tydeliggjøre relieffet. Horisontalplanet er kartet og solhøyden vil da gjelde for hele kartet. Dette er selvfølgelig en gal betraktning i vårt tilfelle, i det solhøyden varierer med bredde – og lengdegrad. Dog vil det være riktig hvis man velger ut et begrenset geografisk område. I Figur 47har jeg puttet inn asimutvinkel=180 (syd) og solhøyden=55 grader som representerer verdier for 1.juli kl.1200 i Oslo(57,95^{o} N 10,72^{o} Ø).

Figur 47: r_jan2005_50_1100_25D_relief



For å lage 3D i GRASS må vi, som allerede nevnt innledningsvis, betrakte nedbørmengdeattributtet som en z koordinat. Det er mulig å utføre i GRASS, men man må utføre flere operasjoner. Først må data eksporteres til ASCII tekstformat, dernest import av data, etter å ha formatert x, y, v (verdi) datasett til x, y og z koordinater. Så langt jeg har sjekket er dette kun mulig med punktgeometrier.

GRASS har støtte applikasjoner for manipulering av 3D rasterdatasett, men i skrivende stund er det mest praktisk hvis nedbørdata allerede er på 3D.
Jeg har allerede "skult" litt til nyere versjoner av GRASS, og vet de har implementert en applikasjon til utvidelse av 2D til 3D.

6.5 Interpolering og statistikk i GRASS

Interpolering er kunsten å beregne ukjente punkter på kurve ut fra kjente punkter, samt kunnskaper eller antagelser om kurveforløpet mellom de kjente punktene [Albregtsen, 2006]

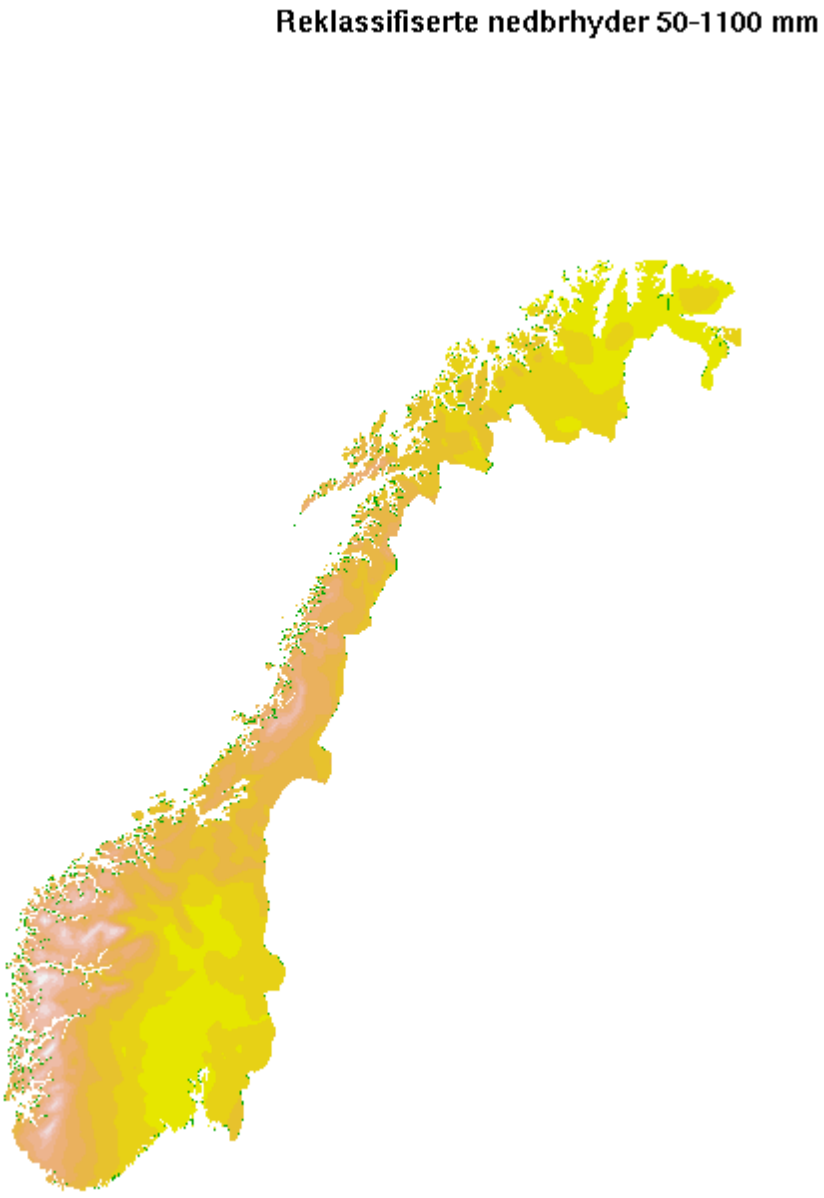
GRASS har innebygde interpoleringsrutiner for overflate og terrenganalyse; spesialiserte rutiner med henblikk på kartproduksjon. Det vi helst ønsker, er generelle og geospatiale statistikk og interpoleringsrutiner for å behandle klimadata, og dernest eksportere dette til

kartdata i GRASS. R – et statistikk språk og åpen kildekode, er et slikt verktøy som kan brukes i GRASS (se http://grass.itc.it/statsgrass/grass_geostats.html). Dette er ikke en innføring i R; se http://www.r-project.org/, samt mengder av andre online læringsdokumenter for R.

Her er et kort interaktivt eksempel i GRASS/R med bruk av eksisterende kartdata.

```
> R

R version 2.4.0 (2006-10-03)
Copyright (C) 2006 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> library(spgrass6)
> G <- gmeta6()
> r_jan2005_50_1100 <- rast.get6("r_jan2005_50_1100")
> r_jan2005_50_1100

Object of class SpatialGridDataFrame
Object of class SpatialGrid
Grid topology:
  cellcentre.offset cellsize cells.dim
1         -303166.5     1000      2205
2         5872771.5     1000      2242
SpatialPoints:
     coords.x1 coords.x2
[1,] -303166.5   5872771
[2,] 1900833.5   8113771
Coordinate Reference System (CRS) arguments: +proj=utm +no_defs +zone=33
+a=6378137 +rf=298.257223563 +towgs84=0.000,0.000,0.000

Data summary:
 r_jan2005_50_1100
 Min.   :   -999.0
 1st Qu.:    100.0
 Median :    200.0
 Mean   :    210.1
 3rd Qu.:    300.0
 Max.   :   1100.0
 NA's   :4623296.0

> image(r_jan2005_50_1100, attr = 1, col = terrain.colors(20))
>  title("Reklassifiserte nedbørhøyder 50-1100 mm")
```

Figur 48: R_r_jan2005_50_1100



Dette eksemplet illustrerer bruk av GRASS spatiale statistikkpakke, eller bibliotek om man vil, som er laget for R og ment å brukes i kombinasjon med GRASS.

I dette eksemplet leser vi inn rasterdata, og lager en visualisering samt legger til en tittel. Dette er ikke mye å skryte av for mitt vedkommende. Det var heller ikke meningen. Det essensielle med å trekke frem R i denne sammenhengen er å henlede oppmerksomheten mot et verktøy for å interpolere nedbørdata, og å lage griddede nedbørdatasett.

6.6 Automatisering av GRASS applikasjoner

Hittil har jeg illustrert hvordan man kan produsere kart fra klimadata, samt i tillegg antydet hvordan vi kan produsere klimadata og overføre dette til GRASS.
Når en kartproduksjon etableres som en rutine, ønsker vi selvfølgelig ikke å utføre tidkrevende enkeltoppgaver. Vi søker å automatisere oppgavene som vi har sett på til nå, og jeg vil her kort presentere to alternativer å gjøre dette på, nemlig via programmering eller via skallskript.

Det enkleste er å lage skallskript. GRASS i seg selv konfigureres via et skallskript `Init.sh` under oppstart. Faktum er at mange av applikasjonen i GRASS er kun skallskript.
Å lage et GRASS skript innebærer å definere essensielle miljøvariable for at GRASS skriptet skal fungere.

Jeg vil her gjengi et eksempel av et skallskript som setter opp essensielle GRASS miljøvariable.
Eksemplet skulle være intuitivt og selvforklarende hvis man har basiskunnskaper i skallprogrammering.

```
#!/bin/sh
echo "LOCATION_NAME: norge_wgs84"    >  $HOME/.grassrc6
echo "MAPSET: rr"                   >> $HOME/.grassrc6
echo "DIGITIZER: none"              >> $HOME/.grassrc6
echo "GISDBASE: /usr/local/share/grass" >> $HOME/.grassrc6
echo "GRASS_GUI: text"              >> $HOME/.grassrc6
# .grassrc6 er en tekstfil som for brukerspesifikk skallmiljøfil, som GRASS
leser.
# Hvis den eksisterer så trenger man ikke de første linjene, såfremt man
# ikke ønsker å endre dem,
# alternativt beholde orginalen og lage en ny ressursfil med annet navn.

export GISBASE=/usr/local/grass-6.0.2
export GISRCRC=$HOME/.grassrc6
export PATH=$PATH:$GISBASE/bin:$GISBASE/scripts

# Prosess ID (PID) som låsefil nummer:
export GIS_LOCK=$$

# Enkel GRASS kommando som gir ut versjonsnummer.
g.version

# Listing av alle rasterdata i mapset "rr" som representerer kun nedbørdata
g.list rast

# For interaktiv input i et ellers automatisert skallskript se
g.ask,g.gisenv, og g.findfile.

# Opprydding
$GISBASE/etc/clean_temp

# Fjern låsefil
rm -rf /tmp/grass6-$USER-$GIS_LOCK

# Slutt på eksempel av et GRASS skallskript
```

I denne sammenhengen er det viktig å nevne at GRASS dermed kan bygges inn i CGI, Perl eller Python, for å nevne noen skriptspråk.

Å programmere med bruk av GRASS-biblioteket er et annet alternativ.
Fordelen med programutvikling er større fleksibilitet og raskere eksekvering av programmet, men kanskje viktigst er muligheten for å lage brukerdefinerte applikasjoner, og dermed utvide funksjonaliteten i GRASS. Et eksempel er applikasjonen `v.out.svg` som tidligere nevnt.

GRASS er utviklet i programmeringsspråket C og det er mange eksempler som er skrevet i C for å studere eller laste ned.
Det er mulig å utvide GRASS funksjonaliteten ved å programmere i andre språk som f. eks C++ og Java for å nevne et par (se http://grass.itc.it/devel/index.php).

6.7 Visualisering i SVG

Fra og med versjon 6.1 av GRASS er det laget en applikasjon `v.out.svg` som eksporterer vektordata til SVG. I skrivende stund opererer jeg med GRASS versjon 6.0.2. Jeg lastet ned kildekoden siden kravet til GRASS versjonsnummer er oppfylt. Applikasjonen krever minimum 6.0.0 versjonen av GRASS. Etter testkjøring var jeg i stand til å visualisere den avledete svg filen i Mozilla Firefox.
SVG modulen er veldig enkel og utelater en mengde attributter som kan gjøre visualiseringen mer attraktiv.
Poenget mitt er ikke at man sitter med en svg modul ”in spe”, men det faktumet at utviklere innen åpen kildekode har som målsetting å utvikle SVG moduler for visualisering. Det er med på å rettferdiggjøre valget av grafikkformatet SVG til visualisering i rammeverket.

6.8 Oppsummering av GRASS

GRASS er et omfattende GIS verktøy i stadig utvikling. Jeg har her sett på kun få, men relevante eksempler på hvordan bearbeide data i GRASS. Jeg har i tillegg gitt en introduksjon til GRASS miljøet, samt hvordan importere og eksportere klimadata inn og ut av GRASS.
Det har i produksjonen vært fokusert på vektordata med tanke på senere visualisering med bruk av åpne standarder.
Videre har jeg også demonstrert hvordan GRASS kan samarbeide med annen åpen kildekode for å utvide funksjonaliteten i behandling av klimadata i statistikk språket R. Jeg har både vist og antydet hvordan man kan automatisere og brukerspesifisere produksjonen i GRASS. Vi har sett på hvordan vi kan utvide funksjonaliteten i GRASS gjennom å programmere og bruk av GRASS-biblioteket.

Geometrisk modellering av rasterdata forenkler algoritmene og reduserer tiden som går med til beregninger, mens en del algoritmer er tilpasset nedbørdata på vektorformatet.
Sluttpoenget må her være at et GIS verktøy må kunne behandle og lagre både vektordata og rasterdata internt, for å kunne tilby flest mulige opsjoner for å transformere nedbørdata til andre formater eller manipulere nedbørdata.

7 PostGIS, en GIS database

PostGIS er en åpen kildekodeløsning og utvidelse til objektsrelasjonsdatabasen PostgreSQL for håndtering av geografiske objekter.
Jeg vil innledningsvis i dette kapittelet gi en generell beskrivelse av hva en GIS database er, og gi en mer fyldig beskrivelse av PostGIS.

GIS-data er digital representasjon av konsepter i den virkelige verdenen (veier, jordbruksområder, høyder, temperatur, nedbør… etc.). En GIS-database inneholder to typer data: Stedfesta (romlige) data og beskrivende data (attributtdata eller egenskapsdata). Til hvert geografiske objekt eller fenomen tilordnes attributter og geometrier som definerer objektets eller fenomenets tilstand. En flate kan f.eks. være en geometrisk representasjon av formen og beliggenheten til en innsjø, skog, tettbebyggelse, nedbørmengde osv.

Attributtdata er enten kvalitative eller kvantitative. Kvalitative data plasserer enhetene i grupper som utelukker hverandre gjensidig.
Det er to typer kvantitative data:

- Data på ordinalnivå rangerer enhetene på en skala. Skog kan f.eks. klassifiseres som høy, middels og lav bonitet (markas evne til å produsere trevirke).
- Data på intervallnivå plasserer enhetene på en skala slik at det kan gjøres beregninger på forskjeller mellom enhetene. Arealer kan f.eks. gis en verdi for årsmiddeltemperatur (se http://no.wikipedia.org/wiki/GIS).

.

7.1 Hva er en GIS database?

En GIS-database bygger som regel på eksisterende databasesystemer som for eksempel hierarkisk database, nettverksdatabase og relasjonsmodellen. Trenden går mot objektorienterte databaser, hvor den geometriske informasjonen og egenskapene er lagret sammen i objekter.
I prinsippet er det ikke noen forskjell å lagre kartdata i forhold til mer vanlige egenskapsdata. Det er ikke så at et system er best egnet for begge strukturer. Egenskapsdata lagres typisk i en eller annen tabellform.

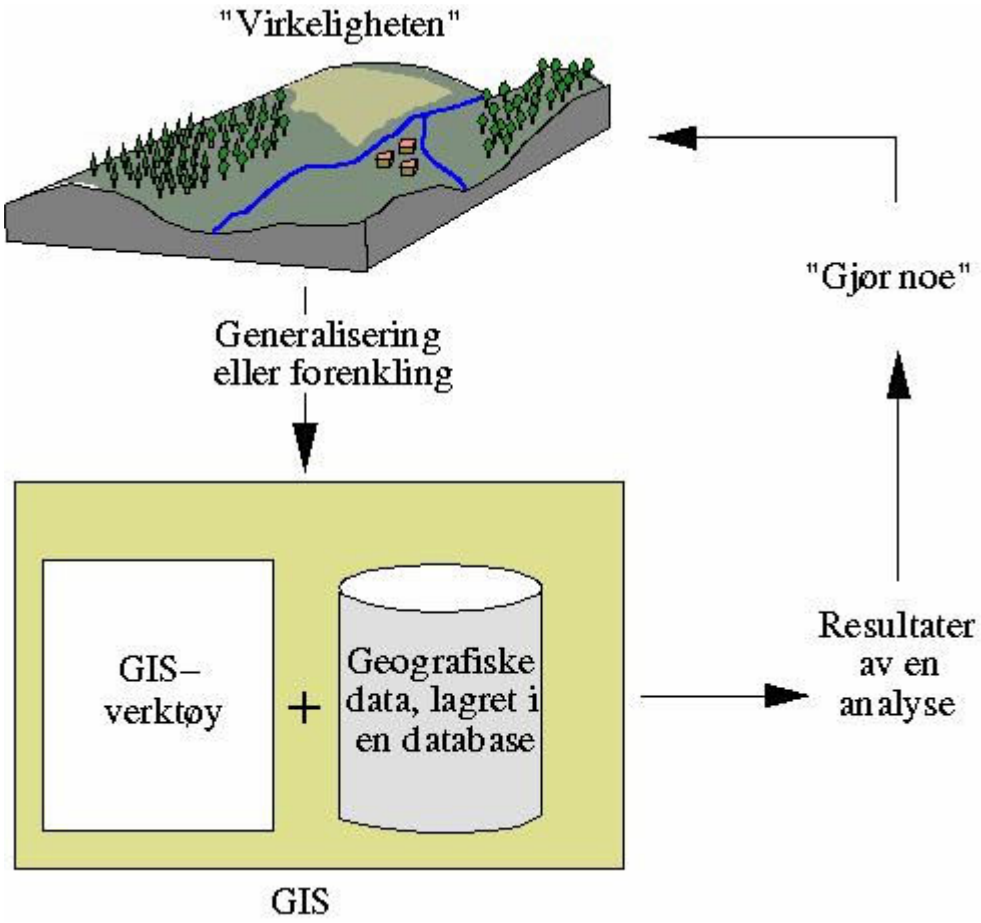
Kommersielle systemer i dag er såkalte *hybride systemer,* hvor geometrien og egenskapene er lagret hver for seg med pekere innimellom. Egenskapene er ofte lagret i relasjonsdatabaser. Det er også slik GRASS strukturerer geometriske data og egenskapsdata på vektorformat.

Objektdatabaser er basert på en mer helhetstenkning av verden rundt oss. Objekter - det være seg mennesker, gjenstander og klimadata assosieres med flere attributter samtidig.
For eksempel attributter som kan assosieres med nedbør er:

- Verdi - mengde i millimeter
- Temporal data - målt på et tidspunkt og gjelder for en tidsperiode
- Type – snø, hagl, regn, yr.
- Spatial data – koordinater på målestasjon der nedbøren ble målt.
- Indirekte spatiale data – stedsnavn, by, postnummer, stasjonsnummer etc.

Ideen med objektdatabaser enkelt forklart er da å gruppere attributter sammen til et ”helhetlig” objekt, som støtter den menneskelige tankegangen om objekter og deres tilstand beskrevet av ett eller flere tilstandsvariable.

Figur 49: Geodatabase [http://www.geo.uio.no/geogr/geomatikk/prosjekt/f-gisinnf/gener.html]



Objektdatabaser er systemer som støtter lagring av datastrukturer som binder attributter sammen, og har et intuitivt brukergrensesnitt. Dette er årsaken til at objektorienterte databaser vinner terreng.

PostgreSQL kan sies å ha en fot i begge leire med å definere seg som en objektrelasjonsdatabase. Det er egentlig kun en utvidelse av relasjonstenkingen. Den støtter objektorientert lagringsstruktur via komplekse objekter, som imidlertid lagres i relasjoner (tabeller).
Utvidelsene blir tilgjengelig for applikasjonen som utvidelser til spørrespråket SQL.
Eksempler på komplekse geometriske verdier i GIS er punkter, linjer og flater, representert i kartesisk eller et geodetisk koordinatsystem.

7.2 Romlige sammenhenger

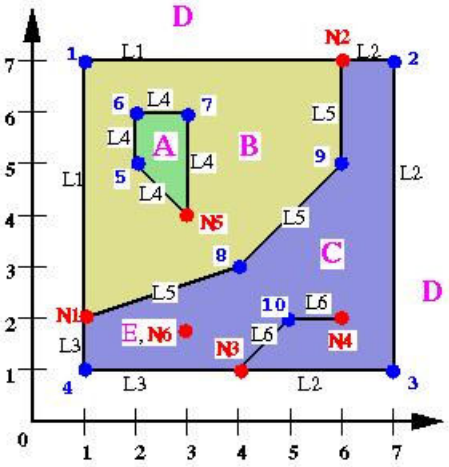
Prosessen som binder egenskapsdata til geografiske koordinater kalles geokoding. Geokoding av geografisk informasjon gjør det mulig å utføre spørringer basert på egenskapsdata og knytte denne informasjonen til et punkt eller et område i terrenget. Med lagring av data for nedbør kunne man for eksempel kjøre en utvalgsspørring i SQL som avleder for eksempel nedbørrike områder.
Generelt i GIS databasesystemer ønsker vi å beskrive romlige attributter utover det å lese av en punktverdi i terrenget.

Romlige attributter i GIS kan være:

- absolutt og relativ lokasjon
- distanse mellom geografiske objekter
- nærhet til geometriske objekter
- geografiske objekter i nabolaget til andre geografiske objekter.
- retning på bevegelse fra plass til plass.
- attributter som resultat av boolske operasjoner ”OG”, ”ELLER", ”DELMENGDE", ”IKKE DELMENGDE”, ”SNITT", "DIFFERANSE" etc.

Topologiske forhold er ofte den mest anvendte sammenhengen som benyttes i en GIS database. Topologi i GIS er en matematisk beskrivelse av hvordan geometrier er sammenkoblet eller ligger i forhold til hverandre. En topologisk datastruktur bestemmer hvor og hvordan punkter og linjer avledes i et kart ved hjelp av knutepunkter (topologiske koblingspunkt). Rekkefølgen på knutepunktene bestemmer avledningen av geometrier. Denne informasjonen er lagret i forskjellige relasjoner i databasen. Ved å lagre denne informasjonen i en logisk sortert rekkefølge, vil man fort kunne oppdage manglende informasjon, f. eks. et manglende linjesegment. Et GIS manipulerer, analyserer og benytter topologisk informasjon for å bestemme hvilke sammenheng det er mellom dataene.
Nettverksanalyse bruker topologisk modellering for å bestemme korteste rute eller alternative ruter fra et sted til et annet.

Figur 50: topologisk datamodell som ligger til grunn for lagring av vektordata [Amlien, Bøhn et al.,1992]



Polygon topologi

Polygon	Linjer
A	L4
B	L1, L4, L5
C	L2, L3, L5, L6
D	utenfor kartet
E	kun et punkt

Node topologi

Node	Linjer
N1	L1, L3, L5
N2	L1, L2, L5
N3	L2, L3, L6
N4	L6
N5	L4
N6	kun et poly

Punkter, Noder Koordinatdata

Punkt, Node	X	Y
N1	1	2
N2	6	7
N3	4	1
N4	6	2
N5	4	3
N6	3	1,5
1	1	7
2	7	7
3	7	1
4	1	1
5	2	5
6	2	6
7	3	6
8	4	2,5
9	6	5
10	5	2

Linje topologi

Linje	Start-node	Slutt-node	Venstre polygon	Høyre polygon
L1	N1	N2	D	B
L2	N2	N3	D	C
L3	N1	N3	C	D
L4	N5	N5	B	A
L5	N1	N2	B	C
L6	N3	N4	C	C

Linjer, punktdata

Linje	Start punkt	Mellomliggende punkt	Slutt punkt
L1	N1	1	N2
L2	N2	2, 3	N3
L3	N1	4	N3
L4	N5	5, 6, 7	N5
L5	N1	8, 9	N2
L6	N3	10	N4

7.3 PostGIS

PostGIS er som allerede nevnt en åpen kildekodeløsning for å håndtere geografiske verdier i PostgreSQL. PostGIS støtter bruken av GiST (Generalized Search Trees) baserte R-Tree spatiale indekser, og funksjoner for analyse og prosessering av GIS objekter.
PostGIS er utviklet av Refractions Research Inc, og er en databaseteknologisk forskningsprosjekt for å forske på håndtering av romlige data. Målsettingen er full støtte for OpenGIS standarder, samt avanserte topologiske konstruksjoner, som dekning, flater og nettverk.
Planen er og har vært å videreutvikle skrivebordsløsninger (eng. desktop) for visning og redigering, samt vevbasert tilgangsverktøy.

"Simple Features Specification for SQL" spesifikasjonen definerer et standard SQL skjema som støtter lagring, uthenting, spørring og oppdatering av geografiske data. Dette er en standard som er implementert i de fleste databasesystemer som støtter geografiske data. Se også http://www.opengis.org/techno/specs/99-049.pdf .

GIS geometrier støttet av PostGIS er et supersett av ”Simple Features” standarden som definert av OGC (OpenGIS Consortium). Fra og med versjon 0.9 støtter PostGIS samtlige geometrier og funksjoner i OGC standarden ” Simple Features for SQL".
PostGIS har egne utvidelser i forhold til standarden for 3DZ, 3DM og 4D koordinater.
3DZ er koordinater definert som XYZ, og er egentlig per definisjon 2.5D
3DM er sann 3D koordinater som XYM hvor M står for mange, er her synonymt med multiple Z verdier over samme XY koordinater.
4D er synonymt med XYZM koordinater.

OGC ”Simple Features” standarden definerer to måter til å representere romlige verdier på. Det ene er Well-Known Text (WKT) formatet og det andre er Well-Known Binary (WKB) formatet.

Eksempler romlige geometrier på tekstbasert WKT format:

- POINT(0 0)
- LINESTRING(0 0,1 1,1 2)
- POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1, 2 1, 2 2, 1 2,1 1))
- MULTIPOINT(0 0,1 2)
- MULTILINESTRING((0 0,1 1,1 2),(2 3,3 2,5 4))
- MULTIPOLYGON(((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1)), ((-1 -1,-1 -2,-2 -2,-2 -1,-1 -1)))
- GEOMETRYCOLLECTION(POINT(2 3), LINESTRING((2 3,3 4)))

OGC standarden krever også at lagringsstrukturen for geometrier også inkluderer en spatial referencing system identifier (SRID).
SRID er påkrevd under opprettelse og innsetting av spatiale objekter i databasen.

OGC standarden støtter i skrivende stund kun 2D geometri, og tilhørende SRID er aldri med i inn- og ut representasjonen av objektdata.
PostGIS utvidede formater er, som tidligere nevnt, et supersett av OGC-standarden. I praksis innebærer det at for hver WKB/WKT finnes det et tilsvarende EWKB/EWKT.
Siden dette ikke er i henhold til OGC-standarden, så er alltid risikoen til stede for at OGC kan komme til å foreslå en tilsvarende utvidelse som er i konflikt med PostGIS sin utvidete standard. Det betyr at man aldri bør innarbeide denne standarden på en slik måte at det vil bli et voldsomt merarbeid å endre dette siden.
Målsettingen til PostGIS er å følge OGC-standarden, selv om man temporært foreslår egne løsninger.
Det har allikevel en akademisk interesse å presentere PostGIS sine egne forslag til utvidelser.
EWKT/EWKB legger til støtter for 3DM, 3DZ, 4D koordinater, samt innebygd SRID informasjon.

Eksempler på tekstbasert representasjon av romlige objekter på EWKT format:

- POINT(0 0 0) – XYZ
- SRID=32632;POINT(0 0) – XY with SRID
- POINTM(0 0 0) – XYM
- POINT(0 0 0 0) – XYZM
- SRID=4326;MULTIPOINTM(0 0 0,1 2 1) – XYM with SRID

- MULTILINESTRING((0 0 0,1 1 0,1 2 1),(2 3 1,3 2 1,5 4 1))
- POLYGON((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0))
- MULTIPOLYGON(((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0)),((-1 -1 0,-1 -2 0,-2 -2 0,-2 -1 0,-1 -1 0)))
- GEOMETRYCOLLECTIONM(POINTM(2 3 9), LINESTRINGM((2 3 4,3 4 5)))

7.4 Bruk av OpenGIS standarder

Opprettelsen av en GIS romlig database i PostgreSQL/PostGIS avledes enten av en GIS mal-database eller via egne PostGIS SQL kommandofiler som eksekveres via psql, som er PostgreSQL sin standard terminalprogram.
Etter opprettelsen etableres to OGC metadata relasjoner SPATIAL_REF_SYS og GEOMETRY_COLUMNS.

SPATIAL_REF_SYS relasjonen innholder numerisk ID og tekstlig beskrivelse av koordinatsystemene brukt i databasen.

SPATIAL_REF_SYS tabelldefinisjon:

```
CREATE TABLE SPATIAL_REF_SYS (
SRID INTEGER NOT NULL PRIMARY KEY,
AUTH_NAME VARCHAR(256),
AUTH_SRID INTEGER,
SRTEXT VARCHAR(2048),
PROJ4TEXT VARCHAR(2048)
)
```

Attributtforklaring:

SRID – unik heltallsverdi som er en ID som identifiserer det romlige referanse systemet i databasen.
AUTH_NAME – Navnet på standarden eller standardiseringsorganisasjonen som er referert for dette referansesystemet. For eksempel EPSG (European Petroleum Survey Group) er en gyldig AUTH_NAME.

AUTH_SRID - ID i det romlige referansesystemet angitt av AUTH_NAME. ID er her EPSG kodetall for kartprojeksjonen.
SRTEXT - Well-Known Text representasjonen av det romlige referansesystemet. Et eksempel er:

```
PROJCS["NAD83 / UTM Zone 10N",
GEOGCS["NAD83",
DATUM["North_American_Datum_1983",
SPHEROID["GRS 1980",6378137,298.257222101]
],
PRIMEM["Greenwich",0],
PostGIS Manual
Paul RAMSEY.
WORKING PAPER
11 / 37
UNIT["degree",0.0174532925199433]
],
```

```
PROJECTION["Transverse_Mercator"],
PARAMETER["latitude_of_origin",0],
PARAMETER["central_meridian",-123],
PARAMETER["scale_factor",0.9996],
PARAMETER["false_easting",500000],
PARAMETER["false_northing",0],
UNIT["metre",1]
]
```

For en full liste av EPSG projeksjonskoder, se http://www.opengis.org/techno/-interop/EPSG2WKT.TXT. For en generell diskusjon av WKT, se OGC "Coordinate Transformation Services Implementation Specification" med URL http://www.opengis.org/techno/specs.htm.

PROJ4TEXT PostGIS bruker Proj4-biblioteket for å utføre koordinattransformasjoner. PROJ4TEXT attributtet inneholder Proj4-koordinatdefinisjon på tekstformat for en definert SRID.
Eksempel: +proj=utm +zone=10 +ellps=clrk66 +datum=NAD27 +units=m.
For mer informasjon om Proj4, se http://www.remotesensing.org/proj.

GEOMETRY_COLUMNS relasjonen er definert som følger:

```
CREATE TABLE GEOMETRY_COLUMNS (
F_TABLE_CATALOG VARCHAR(256) NOT NULL,
F_TABLE_SCHEMA VARCHAR(256) NOT NULL,
F_TABLE_NAME VARCHAR(256) NOT NULL,
F_GEOMETRY_COLUMN VARCHAR(256) NOT NULL,
COORD_DIMENSION INTEGER NOT NULL,
SRID INTEGER NOT NULL,
TYPE VARCHAR(30) NOT NULL
)
```

Attributtforklaring:

F_TABLE_CATALOG, F_TABLE_SCHEMA, F_TABLE_NAME er det fulle, kvalifiserte navnet på objektrelasjonen som inneholder det geometriattributtet. Termene ”skjema” og ”katalog” er Oracle termer. Der er ingen analog attributt ”katalog” i PostgreSQL, så dette attributtet er alltid tomt.
Attributtet ”skjema” eksisterer derimot i PostgreSQL (public er standard skjemaet)
F_GEOMETRY_COLUMN Navnet på geometri attributtet i objektrelasjonen.
COORD_DIMENSION Den romlige dimensjonen (2, 3 or 4 dimensjonal) til attributtet.
SRID er ID for det romlige referansesystemet brukt for koordinatgeometrien i relasjonen. Det er en fremmednøkkelreferanse til SPATIAL_REF_SYS.
TYPE [6] Type romlig objekt. For å begrense det romlige attributtet til en type, brukes en av typene POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION, eller tilsvarende XYM.varianter POINTM, LINESTRINGM, POLYGONM, MULTIPOINTM, MULTILINESTRINGM, MULTIPOLYGONM, GEOMETRYCOLLECTIONM. For en heterogen samling (blanding av flere typer) kan man benytte typen GEOMETRY.

[6] Denne typen er (sannsynligvis) ikke OpenGIS standard, men er påkrevd for å ivareta homogeniteten til GEOMETRY attributtet.

Å opprette en relasjon gjøres i to trinn:

- Opprett en ordinær ikke romlig relasjon.
  Eksempel: `CREATE TABLE ROADS_GEOM (ID int4, NAME varchar(25) )`

- Opprett et romlig attributt til relasjonen med OGC funksjonen `"AddGeometryColumn"`
  Syntaksen er:

  ```
  AddGeometryColumn(<schema_name>, <table_name>,
  <column_name>, <srid>, <type>,
  <dimension>)
  ```

  Eksempel: `SELECT AddGeometryColumn('public', 'roads_geom', 'geom', 423, 'LINESTRING', 2)`

  eller, ved bruk av eksisterende skjema:

  ```
  AddGeometryColumn(<table_name>,
  <column_name>, <srid>, <type>,
  <dimension>)
  ```

  Eksempel: `SELECT AddGeometryColumn( 'roads_geom', 'geom', 423, 'LINESTRING', 2)`

Når den romlige relasjonen er opprettet, kan man begynne å fylle inn GIS data med bruk av SQL INSERT-setninger.

```
BEGIN;
INSERT INTO ROADS_GEOM (ID,GEOM,NAME ) VALUES
(1,GeomFromText('LINESTRING(191232 243118,191108 243242)',-1),'INSERT INTO
ROADS_GEOM (ID,GEOM,NAME ) VALUES (2,GeomFromText('LINESTRING(189141
244158,189265 244817)',-1),'INSERT INTO ROADS_GEOM (ID,GEOM,NAME ) VALUES
(3,GeomFromText('LINESTRING(192783 228138,192612 229814)',-1),'INSERT INTO
ROADS_GEOM (ID,GEOM,NAME ) VALUES (4,GeomFromText('LINESTRING(189412
252431,189631 259122)',-1),'INSERT INTO ROADS_GEOM (ID,GEOM,NAME ) VALUES
(5,GeomFromText('LINESTRING(190131 224148,190871 228134)',-1),'INSERT INTO
ROADS_GEOM (ID,GEOM,NAME ) VALUES (6,GeomFromText('LINESTRING(198231
263418,198213 268322)',-1),'COMMIT;
```

Data kan hentes fra databasen med SQL SELECT-setninger.

```
SELECT id, AsText(geom) AS geom, name FROM ROADS_GEOM;

id | geom | name
---+--------------------------------------+-----------
1 | LINESTRING(191232 243118,191108 243242) | Jeff Rd
2 | LINESTRING(189141 244158,189265 244817) | Geordie Rd
3 | LINESTRING(192783 228138,192612 229814) | Paul St
4 | LINESTRING(189412 252431,189631 259122) | Graeme Ave
5 | LINESTRING(190131 224148,190871 228134) | Phil Tce
6 | LINESTRING(198231 263418,198213 268322) | Dave Cres
7 | LINESTRING(218421 284121,224123 241231) | Chris Way
```

```
(6 rows)
```

Dette er den trivielle fremgangsmåten. Ofte, og kanskje spesielt når man opererer med romlige data, ønsker man å legge inn begrensninger i spørringen for å fokusere inn på interesseområdet.
I forbindelse med romlige datatyper benyttes spesielle operatorer for å legge inn begrensinger i spørringen.

Operatorforklaring:

&& Operatoren forteller om avgrensingsboksen rundt en geometri overlapper med avgrensingsboksen til en annen.

~= Operatoren tester om to geometrier er identisk.. For eksempel, om 'POLYGON((0 0,1 1,1 0,0 0))' er lik 'POLYGON((0 0,1 1,1 0,0 0))' (testen er sann).

= Denne operatoren er litt mer godtroende. Den tester kun om avgrensningsboksene for to geometrier er den samme.

Disse operatorene brukes til å lage betingelser til romlige SELECT-setninger. Når man benytter geometrier og bokser i SQL spørresetningen, må man eksplisitt omforme tekstrepresentasjoner av geometriattributtet ved bruk av "GeomFromText()" funksjonen.

Eksempel:

```
SELECT ID, NAME FROM ROADS_GEOM WHERE GEOM ~=
        GeomFromText('LINESTRING(191232 243118,191108 243242)',-1);
```

Den mest vanlige romlige spørringen vil sannsynligvis være en "rammeorientert" spørring fra en klientapplikasjon som for eksempel nettleser, eller en kartvevapplikasjon, for å ekstrahere kartrammedata for visualisering. Eksempelvis ved bruk av "BOX3D" objektet som ramme kan en SQL spørring se slik ut:

```
SELECT AsText(GEOM) AS GEOM FROM ROADS_GEOM WHERE GEOM
        && SetSRID('BOX3D(191232 243117,191232
        243119)'::box3d,-1);
```

7.5 Bruk av indekser

Indekser gjør det mulig å håndtere store mengder romlige data. Uten indeksering vil hvert søk blitt utført sekvensielt, hvilket innebærer å skanne gjennom hver eneste forekomst i databasen.
Indeksering øker søkehastigheten med å organisere data i en trestruktur som traverseres generelt raskere en sekvensiell skanning, i hvert fall ved store datamengder. For veldig små datamengder vil sekvensielt søk være raskere. PostgreSQL støtter tre standardiserte metoder for indeksering: B-tre, R-tre og GiST indekser.

- B-trær benyttes for å indeksere data som kan sorteres langs en akse; for eksempel, tall, bokstaver, dato. Romlig data kan ikke sorteres langs en akse. Eksempelvis - hva er størst av (0,0), (0,1) eller (1,0)?
- R-trær danner et hierarki av rektangler innenfor rektangler. Punkter som ligger nærmest hverandre vil ligge innenfor et indre rektangel. Vi kan oppfatte at en slik indeks skal kunne betjene spørringer som går på geografisk avstand og lignende. R- tre benyttes i romlige databaser, men R-tre implementasjonen i PostgreSQL er ikke så robust som GiST implementasjonen.
- GiST er en utvidbar datastruktur som tillater indeksering mange forskjellige typer data, inkludert romlige data. "GiST" forener mange av de populære søketrærne i en datastruktur, og dermed eliminerer behovet for å bygge flere typer søketrær for å indeksere forskjellige typer data. PostGIS benytter R-tre indeksering implementert på toppen av GiST for å indeksere romlige data.

Når mengden romlige data overstiger noen tusen instanser, vil man ha behov for å indeksere for å øke søkehastigheten. Hvis søk er basert på GIS attributter (for eksempel nedbørdata som et heltall eller flyttall), vil det i tillegg være behov for å bygge en indeks basert på nedbørdata-attributtet, som kan være et B-tre.

Syntaksen for å bygge en GiST indeks for "geometry" attributtet er som følger:

```
CREATE INDEX [indexname] ON [tablename] USING
        GIST ( [geometryfield] GIST_GEOMETRY_OPS );
```

Å bygge en indeks er en CPU-intensiv øvelse. Avhengig av datamengden og maskinvaren kan tiden for å bygge opp indeksen variere fra sekunder til timer.
Eksempelvis på min Linux server med 2Gb RAM og 2000 MHz CPU, tok det 19 s. å opprette en GiST indeks til en relasjon med ca. 314000 forekomster av punktgeometrier, nedbørverdier med mer i hver forekomst. Dette er ikke en entydig test siden mange faktorer spiller inn, men det gir en grei pekepinn.
Så snart indeksen er opprettet, vil spørreplanleggeren bestemme når den skal bruke indeksen. Spørreplanleggeren i PostgreSQL er dessverre ikke optimalisert for å bruke GiST-indekser i særlig grad. Noen ganger vil derfor søk som ville dra nytte av indekseringen falle tilbake på sekvensiell søk gjennom hele relasjonen.
Det er heldigvis noen tiltak tilgjengelig i PostgreSQL som kan gjør at vi kan komme rundt problemstillingen hvis den oppstår. Man må bare være oppmerksom på at dette kan skje.

7.6 Praktiske eksempler

PostGIS har, som jeg allerede har nevnt, utvidet funksjonaliteten i PostgreSQL med en rekke funksjoner for håndtering av romlige data basert på OGC-standarden "Simple Features Specification for SQL". Det er også funksjoner laget av diverse bidragsytere for ytterligere å utvide PostGIS med ønsket funksjonalitet. De implementeres i PostGIS som egne moduler som lastes ned fra Internett. Et relevant eksempel i mitt arbeid er funksjonene `"AsSVG()"` og `"AsSvgBox()"`.

Syntaksen er som følger:

```
SELECT AsSvg(geometry,[rel],[precision]) FROM <table>
```

- **geom** = geometriattributt av datatypen MULTI(POINT|LINESTRING|POLYGON)
- **rel** = opsjonelt argument for å endre utformat, 0=absolutt eller 1=relativ notasjon
- **precision** = opsjonelt argument for å spesifisere koordinatpresisjon

```
SELECT AsSvgBox(geometry,[rel],[precision]) FROM <table>
```

- **geometry** = geometriattributt
- **rel** = opsjonelt argument for å endre utformat, 0=rect or 1=viewBox notasjon
- **precision** = opsjonelt argument for å spesifisere koordinatpresisjon

Eksempel-relasjon eksportert fra GRASS til PostgreSQL representer nedbørdata som konturkart:

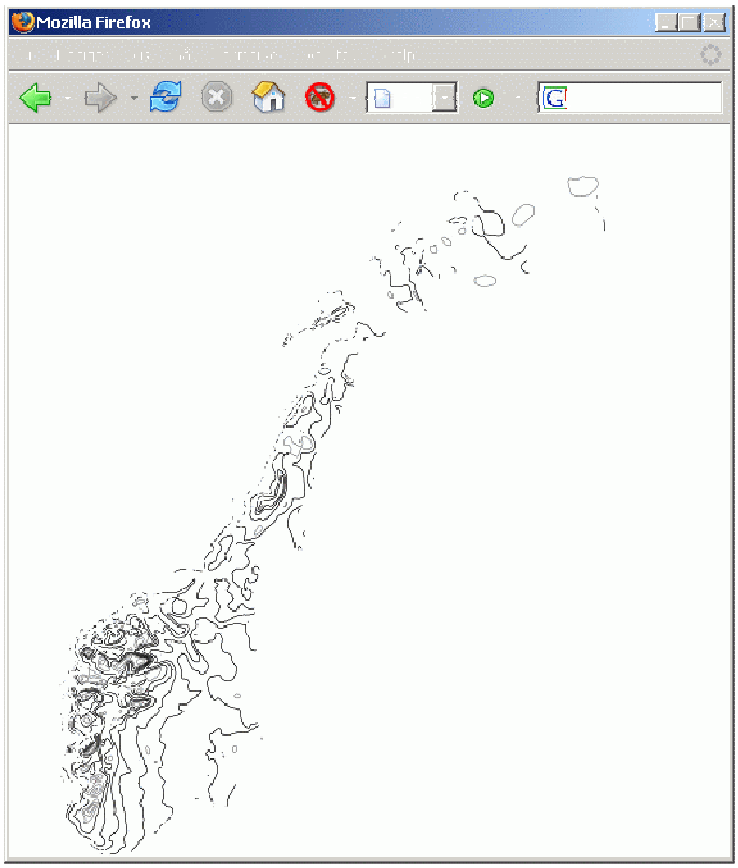
```
CREATE TABLE rr_jan2005_contour
(
  ogc_fid serial NOT NULL,
  wkb_geometry geometry,
  cat int4,
  "level" float8,
  CONSTRAINT enforce_dims_wkb_geometry CHECK (ndims(wkb_geometry) = 3),
  CONSTRAINT enforce_geotype_wkb_geometry CHECK (geometrytype(wkb_geometry)
= 'LINESTRING'::text OR wkb_geometry IS NULL),
  CONSTRAINT enforce_srid_wkb_geometry CHECK (srid(wkb_geometry) = 32767)
)
```

Eksempel på SQL spørring med `AsSvg()`:

```
SELECT '<path d="',assvg(wkb_geometry),'" />' FROM rr_jan2005_contour
```

”AsSvg()” returnerer all SVG geometri som verdien til et SVG <path> element. Derfor må vi trikse litt i SQL spørresetningen for å få med omsluttende SVG <path> merket.
Funksjonen lager altså ikke et SVG dokument basert på utvalgspørringen. Dette gjør vi selv. I mine eksempler har jeg puttet resultatet av utvalgsspørringen mellom det omsluttende <svg> merket, og visualisert resultatet i nettleseren Mozilla Firefox.

Figur 51: PostGIS konturkart utvalgsspørring på SVG format.

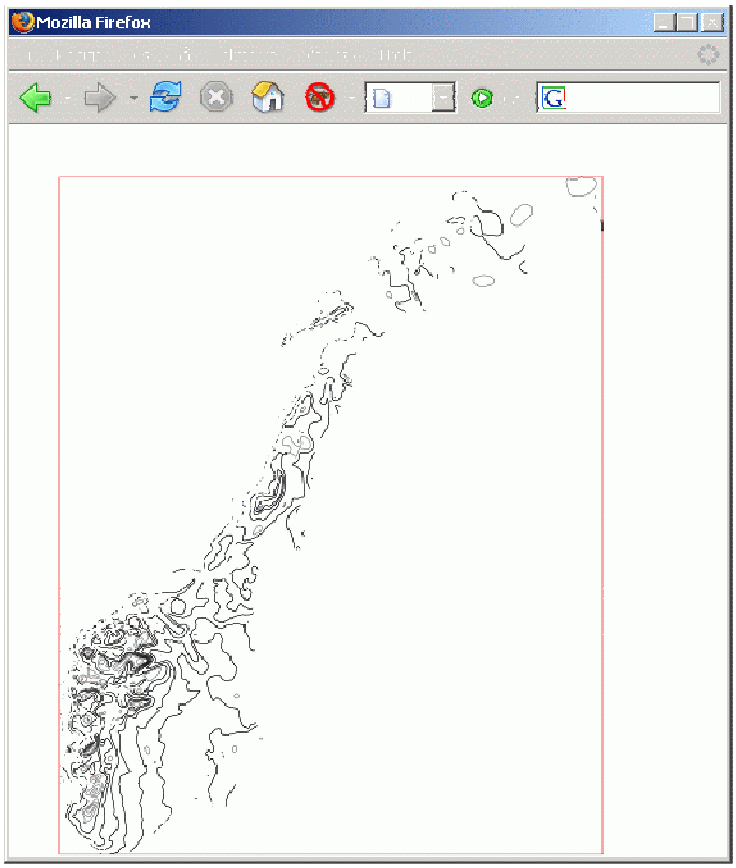


Eksempel på SQL spørring med `AsSvgBox()`:

```
SELECT '<rect ',assvgbox(extent(wkb_geometry),0),'
style="fill:none;stroke:red;stroke-width:1000;" />' FROM rr_jan2005_contour
```

Eksemplet avleder verdien til SVG <rect> merket. Kombineres begge resultatene av begge spørringene i en SVG fil med litt manuell redigering, så får vi Figur 52.

Figur 52: PostGIS AsSvg() + AsSvgBox() eksempel



7.7 Oppsummering

SQL grensesnittet i GRASS er veldig begrenset. Det oppfyller ingen krav til noen av SQL-standardene. SQL-grensenittet brukes i de fleste kjente skriptspråk (Perl, PHP etc…). PostgreSQL er derfor en bedre databaseløsning enn GRASS. Med den romlige utvidelsen PostGIS kan man foreta romlige spørringer og formatere responsdata på XML.

GRASS SQL (i manko av en bedre term) har ikke temporale datatyper. Temporale SQL-spørringer er derfor ikke mulig i GRASS (ikke i den versjonen av GRASS jeg bruker). GRASS SQL-grensesnitt har derimot en bedre responstid grunnet sin enkle databasestruktur.

Rammeverket trenger en bedre databasesystemløsing enn det GRASS tilbyr.
Det meningsløst å konstruere en problemstilling, hvor GRASS og PostgreSQL står mot hverandre. Dette er programvare som supplerer hverandre og er en viktig støtte for rammeverket.

En drøftelse om hvorvidt PostgreSQL er det eneste alternativet som et supplement til GRASS er mer relevant. Jeg har valgt PostgreSQL pga. det er et ”modent” og stabil produkt, i henhold til min og andres erfaring. PostgreSQL er veldig langt fremme når det gjelder utvidet funksjonalitet, eksempelvis utvidelsen PostGIS.

Det skal nevnes at det eksisterer gode alternativer. MySQL (se, http://dev.mysql.com/doc/refman/5.0/en/index.html) og Geodas (se, http://www.ngdc.noaa.gov/mgg/gdas/gx_announce.Html) støtter begge håndtering av romlige datatyper.

8 GIS visualisering av klimadata

Brukergrensesnitt tilpasset klimatologiske kartdata kan utvikles på mange måter og i mange formater. I forhold til mitt forslag til GIS rammeverk befinner vi nå oss hos klienten og vi snakker om prosessering og visualisering på klientsiden. Vi vil gå noe tilbake til tjenersiden når vi skal snakke om datautvelgelse av nedbørdata, men da vil dette bli tydelig fremhevet i teksten.
Figur 1 i kapittel 1, som jeg har referert til ved flere anledninger, illustrerer en type kartbilde kalt koropletkart som er veldig vanlig å benytte for nedbørdata, men også for andre klimadata som f. eks temperatur.

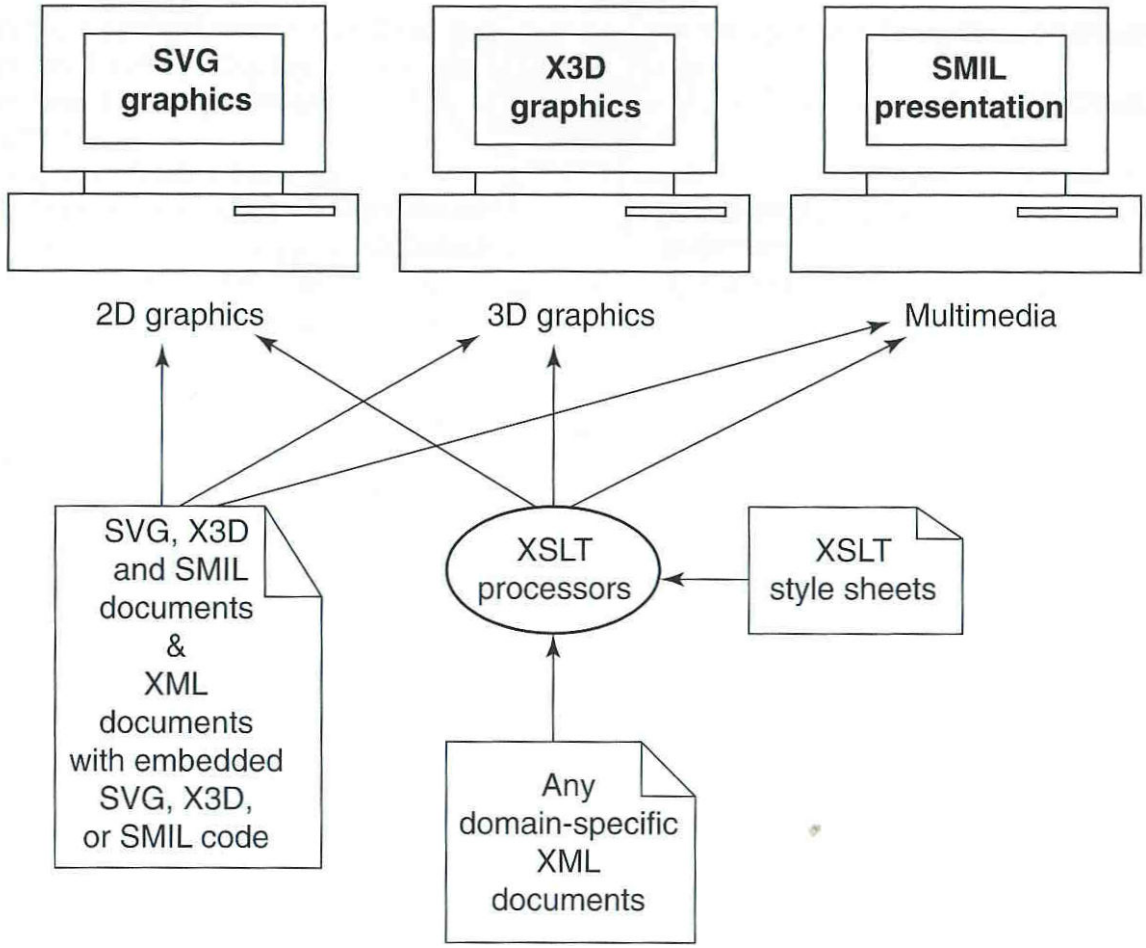
Et kartbilde gjengitt med de riktige proporsjoner i henhold til målestokk er en viktig komponent i visualiseringen.

I tillegg må visualiseringen bestå av tegnforklaringer for nedbørdata, informasjon om målestokk, geografisk orientering og visuelle verktøy for 2D (”3D”) og temporal interoperabilitet.
Det er også ønskelig å kunne hente ned kartdata på rent tekstformat som GML via WFS.
Hva som er et attraktivt grafisk utformet kartbrukergrensesnitt kan være gjenstand for debatt og subjektive meninger, noe som man med selvsyn kan skue i de mange løsninger som finnes på Internett. Dog kan man uttale seg noenlunde objektivt om hvilke visuelle elementer et kartbrukergrensesnitt bør innholde. Så kan man debattere antall zoomnivåer, plassering av målestokk og kompass og grafisk utforming etc. Tegnforklaringer og symboler bør følge konvensjoner som er lett gjenkjennelig for den menneskelige hukommelsen. Dette er noe som er arvet eller utviklet da mange kartbrukergrensesnitt er utformet som en digitalversjon av noe som tidligere var avbildet i bøker/plansjer. Temakart, som for eksempel et klimakart, er også konvensjonelle noe som oftest skyldes at lesere/brukere var fagpersoner. Nå som GIS er noe som blir alt mer og mer brukt, og som er et verktøy som stedfester og tidfester informasjon fra flere fagfelt, og ikke er lenger kun tilgjengelig for fagpersoner, tvinger det parallelt seg frem behov for å standardisere tegnforklaringer og symbolbruk.

Jeg vil i de påfølgende avsnitt drøfte hvordan bruk av SVG i samarbeid med andre nevnte standarder gjør det mulig å lage interoperable klimakart med bruk av ovennevnte kartelementer. Jeg vil i mindre grad fokusere på normer for tegnforklaring og symbolbruk utover det som er vanlig praksis ved Meteorologisk institutt. Jeg kan nevne at der pågår samarbeid med andre instanser for å utvikle en felles GIS portal. Et eksempel på et samarbeid for å utvikle en GIS portal er ”Norge digitalt” (se http://www.statkart.no/IPS/?template=norgedigitalt).

Jeg vil avslutte denne innledningen med å vise en skjematisk fremstilling av visualisering av XML data. Se Figur 53.

Figur 53: visualisering av xml data [Geroimenko, 2005]



8.1 Begrepsforklaringer

Det er noen sentrale begreper forbundet med digitale kartbilder og visualisering, noen er innlysende, mens andre trenger en forklaring. Jeg har bevisst valg norske ord som erstatning for populære engelske termer. Forklaringene her danner grunnlaget for bruk av begrepene utover i dette kapittelet. Forklaringene er hentet fra Statkart sin "Ordbok for kart og oppmåling" (se http://www.statkart.no/IPS/filestore/cd2003/std/oko/oko.pdf).

- Kartbildekoordinatsystem – fritt valgt koordinatsystem som beskriver posisjonen av punkter i bildet - vanlig brukt er et rettvinklet todimensjonalt kartbilde med akser i bildeplanet og origo definert på grunnlag av avbildede rammemerker, eller et rettvinklet tredimensjonalt kartbilde med origo i projeksjonssentret og Z-aksen vinkelrett på bildeplanet. X velges vanligvis langs basis og Y vinkelrett på X.
- Koordinattransformasjon – matematisk eller grafisk overføring av punkter fra ett koordinatsystem til et annet etter entydige matematiske regler, transformasjonsformler) ved f.eks. målestokkendring, rotasjon og forflytning (translasjon); jf. konform transformasjon, affin transformasjon, absolutt orientering
- Grafisk utforming (eng. layout) eller utlegg.
- Tegnforklaring (eng. legend) – er en sammenstilling og forklaring av symbolene som er brukt ved kartfremstillingen.

- Panering (eng. panning) – beskriver interaksjonen med mus og musepeker ved å låse/feste musepeker til kartbildet for å dra/trekke/forskyve det i ønsket retning.
- Piksel (eng. pixel) er et bildeelement; minste element i et bilde, f.eks. et punkt i en dataskjerm.
- Relativ koordinat – koordinatverdi i et koordinatsystem som selv er spesifisert relativt til et annet koordinatsystem.
- Vindu (eng. window) omriss, oftest rektangulært, som avgrenser den delen av data en ønsker å behandle, f.eks. for fremvisning på en dataskjerm eller tegnemaskin.
- Forstørring/forminsking (eng. zoom (in/out)) – dvs. endring av målestokk.

8.2 Brukergrensesnittproblemstillinger

Vi kan dele problemstilling med utvikling av brukergrensesnitt inn i tre:

- Grafisk utforming
- Datautvelgelse
- Interoperabilitet

Jeg vil i påfølgende avsnitt drøfte kulepunktene og bruk av SVG standarden (og til slutt litt om X3D).
De fleste er kjent med standardene HTML og XHTML som er mye brukt for å lage vevsider. HTML og XHTML kan innholde andre dokumenter på andre formater, deriblant SVG. HTML og XHTML er markeringsspråk som benyttes til grafisk utforming av vevsider. I dette kapitlet vil fokus være på å drøfte hvordan man kan utvikle brukergrensesnittet basert på SVG og X3D standarden. Hensikten min er å gi en innføring i SVG og litt om X3D standarden som omfatter mer enn primitive geometriske objekter i 2D og 3D.

8.3 Fordeler med bruk av SVG og X3D standardene

Det er mange fordeler med bruk av XML standardene SVG og X3D.

- Høy kvalitets vevgrafikk med presis strukturell og visuell kontroll
- Dynamisk generert grafikk drevet av sanntids XML data.
- Interaktiv og skript-utviklet grafikk med bruk av XSLT, Javaskript, DOM, Java, Perl, Visual Basic med mer.
- Grafikk som er bundet til alle språk som er medlemmer av XML familien.
- Personlig preferanser til grafikken realisert gjennom effektive metoder for brukerspesifisering.
- Grafikk som automatisk kan optimaliseres for mobile enheter som for eksempel mobiltelefon og PDA (personlig digital assistent).
- Effektiv oppdatering av grafikk (siden design er atskilt fra data).
- Brukerespesifisert zooming av grafikk for å øke/minske detaljeringsgraden.
- Støtte for internasjonalisering og nasjonalisering av språk og enheter.
- Grafikken på lesbart tekstformat.
- Åpen og royalty fri standard.

Siden SVG og X3D er XML-språk kan de integreres med hverandre. SVG kan for eksempel utvides til 3D. Teoretisk vil en blanding av SVG- og X3D-syntaks åpne for nye spennende muligheter og nyttige ”alt i ett” applikasjoner.

SVG og X3D er de to gjeldende standardene for vevgrafikk, standardene vil spille en større og større rolle for å visualisere informasjon og utvikling av grafiske brukergrensesnitt i neste generasjons vevgrafikk. De representerer valgt standard for XML-basert grafikk i den fremtidig XML-språklig dominante verdensveven. Standardene støtter dynamisk visualisering av XML-dokument, er og tett og sømløs kobling med andre tallrike XML-baserte teknologier og bruk av metadata, ontologi og annen semantisk vevteknologi.

8.4 Mer spesifikt om SVG standarden

SVG viderefører de underliggende premissen til HTML som med et deklarativt språk kan beskrive en omfattende subsett av alle underliggende hyperlenkede dokumenter. SVG viderefører dette konseptet for å beskrive dokumenter basert på vektorgrafikk. Resultatet er et språk som er mye mer enn bare et grafikkformat. I følgende kulepunkter ramser jeg opp egenskaper iboende i SVG standarden.

- **Navngitte grafiske entiteter**. Alt i SVG kan gis et navn og kan som følge av dette refereres. Dette skiller seg fra andre språk for vektorgrafikk som tillater navnegrupper eller lag, men som sjelden gir mulighet for disse til å manipulerer individuelle enheter.
- **Animasjon**. Animasjon innebærer temporal endring av deler av grafikken. Endring kan være endring i posisjon, fyllfarge, tekst etc. SVG støtter SMIL (Synchronized Multimedia Integration Language) som gir mulighet for kontroll over forskjellige grafiske egenskaper og definisjon av stier (path), lyd og video.
- **Tekst**. Muligheten til å jobbe med tekst på linjebasis og glyfer (eng. glyph). En glyf er en visualisering av et tegn. Eksempelvis er A **A** *A* tre ulike glyfer for det samme tegnet. SVG støtter også innbygget HTML kode. Neste generasjon SVG standard vil støtte bygging av flytdiagrammer og sider.
- **Interaktiv programmering**. Som HTML, støtter SVG DOM (Document Object Model) som er abstrakt programmeringsspråk-grensesnitt for manipulering av SVG grafikk. Selv om Javaskript kanskje er det mest benyttede programmeringsspråket, finnes det SVG prosessorer som kan jobbe med Java, C++, Perl og PHP.
- **Lenkede ressurser**. SVG håndterer ressurser ved å lenke grafikk eller grafikkelementer (som for eksempel fyll) til annen grafikk. Med ressurser forstås også andre SVG elementer (også eksterne SVG dokumenter) som muliggjør utvikling av biblioteker og klassefunksjoner for grafikk.
- **Metadata**. Siden SVG er XML-basert, er det mulig å inkludere metadata om spesifikke elementer i grafikken samt å binde disse metadata til eksterne metadata. Dette gjør SVG mer selvforklarende, selv uten bruk av interaktive operasjoner.

Bruk av beskrivende grafikk, temporal og hendelsesstyrt animasjon og metadata gjør SVG ganske unik blant eksisterende grafikkspråk. Disse egenskapene er generelt ofte forbundet med et fullt brukergrensesnitt API (Application programming interface) som for eksempel Microsofts DirectX og GDI arkitekturen i Linux. Det at SVG i tillegg er XML-basert, gir SVG en fordel som andre deskriptive språk, som f. eksempel Postscript, ikke kan konkurrere med.

Siden SVG er et XML-språklig dokument, kan et signifikant antall manipulasjoner av et SVG dokument utføres ved bruk av det generelle XML DOM-grensenittet i stedet for å benytte et separat API utviklet for manipulering av SVG teknologien. XML DOM-grensesnittet benyttes på alle typer XML dokument. En ytterligere fordel med bruk av XML DOM betyr implisitt at gjeldende navnerom (eng. namespace) i et SVG dokument også kan manipuleres. Med dette menes at i stedet for å manipulere XML elementer i navnerom basert på spesifikke definerte konvensjoner for manipulasjon, vil XML DOM manipulasjoner gjelde i alle nivåer og reduserer behovet separate klasser for at XML språket skal fungere.

Definisjon av XML navnerom:

Et navnerom er deklarert ved å bruke en familie av retjenerte egenskaper. Navnene til slike egenskaper må enten være xmlns eller ha xmlns: som et prefiks. Disse egenskapene, slik som en hvilken som helst annen XML egenskap, kan bli gjort tilgjengelig direkte eller som en standard verdi.

Bruk av XSLT kan for eksempel transformere et XML dokument som inneholder kun klimadata til et grafisk format som f. eks et temakart eller kartesisk kurvediagram representasjon i SVG med bruk av sjablongfiler som beskriver transformasjonen og er helt uavhengig av den grafiske kodingen. Det innebærer at selve transformasjonen kan skje på en tjener, mens visualiseringen skjer kun hos klienten.

8.5 Grafisk utforming i SVG

I dette avsnittet skal vi se nærmere på den strukturelle oppbyggingen av et SVG dokument. Fokus er på SVG elementer som kan kobles opp mot relevante visualiseringer av kart, kartforklaringer samt kartverktøy. Se http://www.w3.org/Graphics/SVG/ for detaljert informasjon om SVG-standarden.

SVG er, som allerede nevnt i kapittel 1.6, et XML-basert markeringsspråk og er dermed underordnet XML syntaks. Les mer om XML-standarden på vevsidene hos W3C (http://www.w3.org/TR/xml/). Før jeg går videre, vil jeg gjerne presisere at i et XML-basert markeringsspråk har alle elementer et startmerke og sluttmerke – altså <svg> og </svg>. Jeg benytter kun startmerket når jeg referer til et element utover i teksten.
Alle XML-dokument må ha et rot element, og i et SVG dokument er dette markert med merket <svg>. Dette elementet har en rekke roller. Et SVG dokument kan ha flere underordnede <svg> merker. En annen måte å uttrykke det samme er å si at vi kan ha nøstede <svg> elementer. En viktig rolle for <svg> merket er deklarasjon av fundamentale navnerom, blant annet svg navnerommet selv som er:

```
xmlns:svg=" http://www.w3.org/2000/svg"
```

og det mye brukte navnerommet til xlink:

```
xmlns:svg="http://www.w3.org/1999/xlink"
```

Et viktig konsept i forbindelse SVG, som også er helt essensielt for kartvisualisering, er koordinatsystemet. I denne forbindelsen er det et par begreper som må forstås. Begrepet kanvas (eng. canvas) er i SVG beskrevet som flaten hvor grafiske elementer er avtegnet. Det kan være et fysisk medium som papir eller pc-skjerm, eller mer abstrakt et adressert minneområde i en PC. Kanvasen er området i SVG hvor alt visualiseres, og er i prinsippet et uendelig stort todimensjonalt område. Visualiseringen på kanvasen utføres i et definert rektangulært område på engelsk kalt Viewport - fritt oversatt til utsynsportal. Koordinatsystemet som er definert for utsynsportalen er også referansen for brukerkoordinatsystemet. I utgangspunktet er disse like (se Figur 54).

Figur 54: SVG koordinatsystem [http://www.w3.org/TR/SVG/coords.html#InitialCoordinateSystem]

Utsynsportalens utstrekning (bredde og høyde) er oppgitt i målenheter som for eksempel px, pt, mm, in og prosent. Plassering er definert av x og y elementattributtene som implisitt har verdien 0 hvis ikke noe verdi er definert. Brukerkoordinatsystemet derimot er gitt med tall uten måleenheter, noe man kanskje ikke hadde forventet. Brukerkoordinatsystemet er definert med kun tallstørrelser i x, y samt bredde og høyde i forhold til den fysiske størrelsen på utsynsportalen. Dette er essensielt å kjenne til når vi ønsker å gjengi kart i riktige proporsjoner.

Her er SVG dokumentet for Figur 54:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="300px" height="100px" version="1.1"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Example InitialCoords - SVG's initial coordinate system</desc>
  <g fill="none" stroke="black" stroke-width="3" >
    <line x1="0" y1="1.5" x2="300" y2="1.5" />
    <line x1="1.5" y1="0" x2="1.5" y2="100" />
  </g>
  <g fill="red" stroke="none" >
    <rect x="0" y="0" width="3" height="3" />
    <rect x="297" y="0" width="3" height="3" />
    <rect x="0" y="97" width="3" height="3" />
  </g>
  <g font-size="14" font-family="Verdana" >
    <text x="10" y="20">(0,0)</text>
    <text x="240" y="20">(300,0)</text>
    <text x="10" y="90">(0,100)</text>
  </g>
</svg>
```

I koden over skal man innse at utsynsportalen og brukerkoordinatsystemet er like. Dette kan man endre på som vist i følgende kodesnutt:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="300px" height="100px" viewBox="0 0 600 200" version="1.1"
     xmlns="http://www.w3.org/2000/svg">
```

Brukerkoordinatsystemet er definert med attributtet ”viewBox ”og doblet skalering, noen som halverer størrelsen på Figur 54, men har samme utgangspunkt.

”viewBox” attributtet:

```
 viewBox="min-x min-y bredde høyde"
```

Det som er verdt å merke seg her, er at y koordinater har økende verdier ”nedover” på en PC-skjerm. Det betyr at en direkte innsetting av kartdata med gitte koordinater for bredde- og lengdegrad vil få kartet til å stå opp ned. Dette løses ved å multiplisere y koordinaten (breddegraden) med -1. Så sant ”viewBox” ikke er definert, vil koordinatsystemet være definert av det ”ytterste” svg elementet, og vil gjelde for hele SVG dokumentet. Utgangspunktet for ”viewBox” kan også endres med å sette andre verdier enn 0 0 for ”min-x min-y”.
På samme måte kan man endre x og y verdiene i nøstede <svg> elementer. Dette er gjør det mulig å operere med flere koordinatsystemer og plasseringer av elementer relativt til det ytre koordinatsystemet definert i det ytterste <svg> beholderelementet. Dette er helt essensielt for å kunne lage en grafisk kartutforming som inneholder forskjellig type informasjon som nevnt tidligere i dette kapittelet. Dette blir tydeligere eksemplifisert utover i dette kapittelet.

Først vil jeg presentere flere relevante elementer i SVG standarden.
Elementet <defs> definerer en seksjon i SVG-dokumentet som siden kan refereres av <use>-uttrykket. Dette gjør det mulig å bygge opp et symbolbibliotek som siden kan brukes/refereres i andre deler av et SVG dokument med å bruke et <svg> og dets ”id” attributt.
Attributter til elementer eksplisitt definert inne i en slik beholder eller seksjon er å betrakte som invarianter. Det betyr at et attributt som er definert for et element i <defs> seksjonen kan siden ikke endres. Ved å la være å definere et attributt, som skal kunne endres siden, i denne seksjonen, oppheves denne restriksjonen hvis hensikten ikke er å ha attributter som invarianter.

Et annet element som er et ”dynamisk” alternativ til <defs> er <symbol>. Dette elementet er mer likt <svg> med den forskjellen at alt innhold i <svg> automatisk avtegnes mens visualisering av innholdet i <symbol> skjer kun ved eksplisitt bruk av <use>. Dette gjør det ideelt å bruke <symbol> for visuelle elementer man ønsker skal være mer fleksible dvs. motsatt effekt av elementer som er definert i en <defs> elementbeholder.
Eksempler på kartelementer som typisk kan plasseres i <defs> eller <symbol> elementbeholderen er kartmålestokk, tegnforklaringer med mer. Dette er typiske kartelementer man ønsker å bruke om igjen i andre settinger og kan vær dynamisk eller statisk, alt etter applikasjonen intensjon.

Et viktig element i SVG er <g> elementet som i standarden er referert til som gruppe elementer. En alternativ tolkning kunne gjerne vært ”grafisk kontekst”. Dette elementet har tre distinkte anvendingsområder.

- **Gruppere relaterte elementer sammen**. Dette er den mest vanlige bruken. <g> elementet fungerer som en parentes rundt elementer og binder dem sammen til en kohesiv enhet. <g> elementet kan ha sin egen ID som siden kan refereres med <use> eller DOM. Jeg kommer tilbake til DOM og andre interaktive verktøy siden i sammenheng med SVG dokumenter.
- **Hendelser/CSS håndtering**. Tilordning av et CSS attributt (se senere i avsnittet for beskrivelse av <text> elementet) vil i praksis tilordne dette attributtet til alle elementer som er lagret i gruppe beholderen. Dette er en egenskap som er viktig i forbindelse med å utvikle interaktive SVG-dokumenter - mer detaljer om dette i neste avsnitt. Kort fortalt - hendelsesfunksjoner plassert i gruppeelementet <g> vil analogt med det jeg skrev om attributter kunne påvirke alle elementer inne i <g> elementet.
- **Transformasjoner**. Det er mulig å utføre matrise (affine) transformasjoner på et <g> element, dvs. forskyve koordinatsystemet.

Jeg vil si litt mer om affine transformasjoner da det er et eget attributt og viktig konsept i SVG-standarden. Tekst og formel er hentet fra http://www.ifi.uio.no/~inf2310/v2007/20070227_geometriskeOperasjoner.pdf.

Affine transformasjoner er å transformere koordinatene (x, y) til (x’, y’).

Affine transformasjoner kan beskrives med likningene:

```
x’= a0x+a1y+a2

y’= b0x+b1y+b2
```

Figur 55: matriseformen

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_0 & a_1 & a_2 \\ b_0 & b_1 & b_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \text{eller} \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_0 & a_1 \\ b_0 & b_1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a_2 \\ b_2 \end{bmatrix}$$

Eksempler på affine transformasjoner er skalering, rotasjon, ”shearing”, translasjon og kombinasjoner av disse.
I SVG utføres affine transformasjoner ved tilordning av `transform` attributtet som er definert for en rekke SVG-elementer.

Intuitivt forstår man hva som kan utføres på elementer med slike operasjoner. I et kartdokument vil translasjon av kartdata være synonymt med panering, altså det å dra eller skyve på et kart i alle retninger. På samme måte kan vi tenke oss rotasjon av et kartobjekt om en av aksene.

Jeg har tidligere nevnt bruk av brukerkoordinatsystem i SVG. En eksplisitt translasjon er nettopp å definere x og y i et brukerkoordinatsystem uten bruk av *transform* attributtet.

SVG har definert en rekke basale geometrielementer som sirkel, ellipse, rektangel, linje, polygon og sti (eng. path). <path> er det mest komplette elementet, fordi det kan brukes til å lage alle basale geometrier. GRASS og PostGIS vil en SVG formatering oftest bruke <path> elementet som formatet for geometrier.

Til nå har vi drøftet typiske grafiske elementer for grafisk utforming av et SVG kartdokument. SVG gir imidlertid mye mer enn visualisering av basale geometriske objekter.

En betydningsfull side ved SVG er muligheten for å lagre kontekst på alle nivåer i et SVG dokument. I et kartdokument kan dette typisk være informasjon som forteller hva dette kartdokumentet skal brukes til. Elementene <title>, <desc> og <metadata> er til for å lagre kontekst på forskjellige nivå i et SVG kartdokument. <title> er kanskje det enkleste og mest elementære elementet av disse tre. Med tekstlig innhold brukes <title> til å sette en merkelapp på grafikken og/eller subsidiær grafikk. Forslag til bruk kan for eksempel være å vise verktøytips eller annen informasjon ved å bevege musen over grafikken. Eller den mer intuitive bruken er å sette tittelen på grafikken hos klienten. SVG standarden sier ingenting om hvordan dette elementet skal brukes, men gir i stedet forslag til bruk for å gi utviklere av klientprogramvare en ide om intensjonen til dette elementet.

I <desc> deskriptivt element er intensjonen å lagre mer kompleks informasjon. Det kan for eksempel inneholde elementer fra andre markeringsspråk. Et scenario er å benytte XML kode inne i elementet sammen med kontekst for kartgrafikken. Hvis SVG kartdokumentet ble innebygd i et HTML/XHTML dokument kunne klientapplikasjonen laste ned og visualisere denne informasjonen. Her kunne man for eksempel beskrive at kartdokumentet inneholder nedbørdata for en gitt tidsperiode, maksimums- og minimumsnedbørstatistikk samt geografisk ustrekning på kartet med mer. Slik informasjon kunne da kodes i henhold til ønsket visualisering i måldokumentet. Koding og semantikk er bestemt av utvikleren av kartdokumentet.
Til sist <metadata > som har mange likhetstrekk med <desc>, men ikke er identisk. Innenfor dette elementet er koding og semantikk bestemt av definerte metadata-språk, som for eksempel Resource Description Framework (RDF, se også http://www.w3.org/RDF/). RDF benytter et sett med merker i språket, disse kan leses av et program eller programtillegg som kan tolke dette.

Her følger et eksempel fra W3C som inkluderer alle tre elementene:

```
<?xml version="1.0" standalone="yes"?>
<svg width="4in" height="3in" version="1.1"
    xmlns = 'http://www.w3.org/2000/svg'>
    <desc xmlns:myfoo="http://example.org/myfoo">
      <myfoo:title>This is a financial report</myfoo:title>
      <myfoo:descr>The global description uses markup from the
        <myfoo:emph>myfoo</myfoo:emph> namespace.</myfoo:descr>
      <myfoo:scene><myfoo:what>widget $growth</myfoo:what>
      <myfoo:contains>$three $graph-bar</myfoo:contains>
        <myfoo:when>1998 $through 2000</myfoo:when> </myfoo:scene>
   </desc>
    <metadata>
      <rdf:RDF
           xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

```
          xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
          xmlns:dc = "http://purl.org/dc/elements/1.1/" >
        <rdf:Description about="http://example.org/myfoo"
             dc:title="MyFoo Financial Report"
             dc:description="$three $bar $thousands $dollars $from 1998
$through 2000"
             dc:publisher="Example Organization"
             dc:date="2000-04-11"
             dc:format="image/svg+xml"
             dc:language="en" >
          <dc:creator>
            <rdf:Bag>
              <rdf:li>Irving Bird</rdf:li>
              <rdf:li>Mary Lambert</rdf:li>
            </rdf:Bag>
          </dc:creator>
        </rdf:Description>
      </rdf:RDF>
    </metadata>
</svg>
```

Eksemplet (se http://www.w3.org/TR/SVG/metadata.html#MetadataElement) beskriver en finansrapport, men en relevant beskrivelse kunne være en beskrivelse av nedbørstatistikken for januar 2005 som et eksempel. Eksemplet illustrerer hvilke mangfoldige muligheter det er å lagre informasjon i et SVG-dokument. Denne informasjonen har vi sett kan leses direkte i et klientprogram, men kanskje mer viktig er muligheten for å overføre kodet informasjon som kan fortolkes av et program eller programtillegg. I et kartdokument er det informasjon som kan struktureres på forskjellige nivå.

Vi skal se mer på tekst i SVG. Tekst-elementet i SVG merkes <text>. Elementet tjener som en beholder for å karakterisere tekst og/eller for å spesifisere at innholdet er tekstdata. Tekstdata kan være en sekvens av tegn og/eller CDATA separerte tegn, og som en aggregering av flere tekstlige elementer som for eksempel <tspan>, <tref> og <textPath>. <tspan> gjør koordinattransformasjon av tekst mulig, mens <tref> gjør det mulig å referere til tekst definert et annet sted. CDATA i XML og SVG er elementer som ikke skal fortolkes. <textPath> gjør det mulig å skrive en tekst langs en annen sti enn en rett linje. Et lite omgjort eksempel fra W3C illustrerer dette bedre enn ord.

```
.
.

  <defs>
    <path id="minsti"
          d="M 100 200
             C 200 100 300   0 400 100
             C 500 200 600 300 700 200
             C 800 100 900 100 900 100" />
  </defs>
  <desc>Eksempel fra W3C – tekst langs buktende sti</desc>
  <use xlink:href="#minsti" fill="none" stroke="red"  />
  <text font-family="Verdana" font-size="42.5" fill="blue" >
    <textPath xlink:href="#minsti">
      Alt som går opp, må komme ned, alt som går..
    </textPath>
  </text>
  <!-- visning av randen på kanvas med bruk av 'rect' element -->
```

```
  <rect x="1" y="1" width="998" height="298"
        fill="none" stroke="blue" stroke-width="2" />
.
.
```

Figur 56: <textPath> visualisering av kodesnutt

All tekst i SVG benytter Cascading Style Sheet (CSS) tekstattributter, i tillegg til et knippe egendefinerte tekstattributter.

Et av de mest fascinerende aspekter med SVG er at definering av fonter simpelthen er en sti <path> definisjon. I SVG kan man definere egne fonter og sett med fonter. SVG kaller egendefinerte fontelementer for <glyph> glyfer, som jeg nevnte tidligere i dette avsnittet. Sett i forhold til <symbol> kan <glyph> kun defineres i kontekst med fonter.
Et lite <glyph> eksempel følger (se http://commons.wikimedia.org/wiki/Image:Part_of_glyph_-_counter.svg):

```
.
<defs>
  <style type="text/css">
   <![CDATA[
    .str0 {stroke:#1F1A17;stroke-width:4}
    .fil0 {fill:none}
    .fil1 {fill:#666666}
    .fil2 {fill:#FF3300}
   ]]>
  </style>
 </defs>
 <g id="Layer 1">
  <path class="fil0 str0" d="M0 29c38,0 387,0 387,0"/>
  <path class="fil0 str0" d="M0 132c38,0 387,0 387,0"/>
  <path class="fil0 str0" d="M0 4c38,0 387,0 387,0"/>
  <path class="fil0 str0" d="M0 310c38,0 387,0 387,0"/>
  <path class="fil1" d="M371 220c0,30 -8,53 -23,69 -…

  <path class="fil1" d="M175 283c3,1 5,3 6,7 0,0 0,0 0,1…
.
.
  <path class="fil2" d="M113 268l0 -45c0,-3 -1,-5 -3,-…
.
.
  <path class="fil2" d="M335 223c0,-22 -5,-40 -16,-55 …
 </g>
</svg>
```

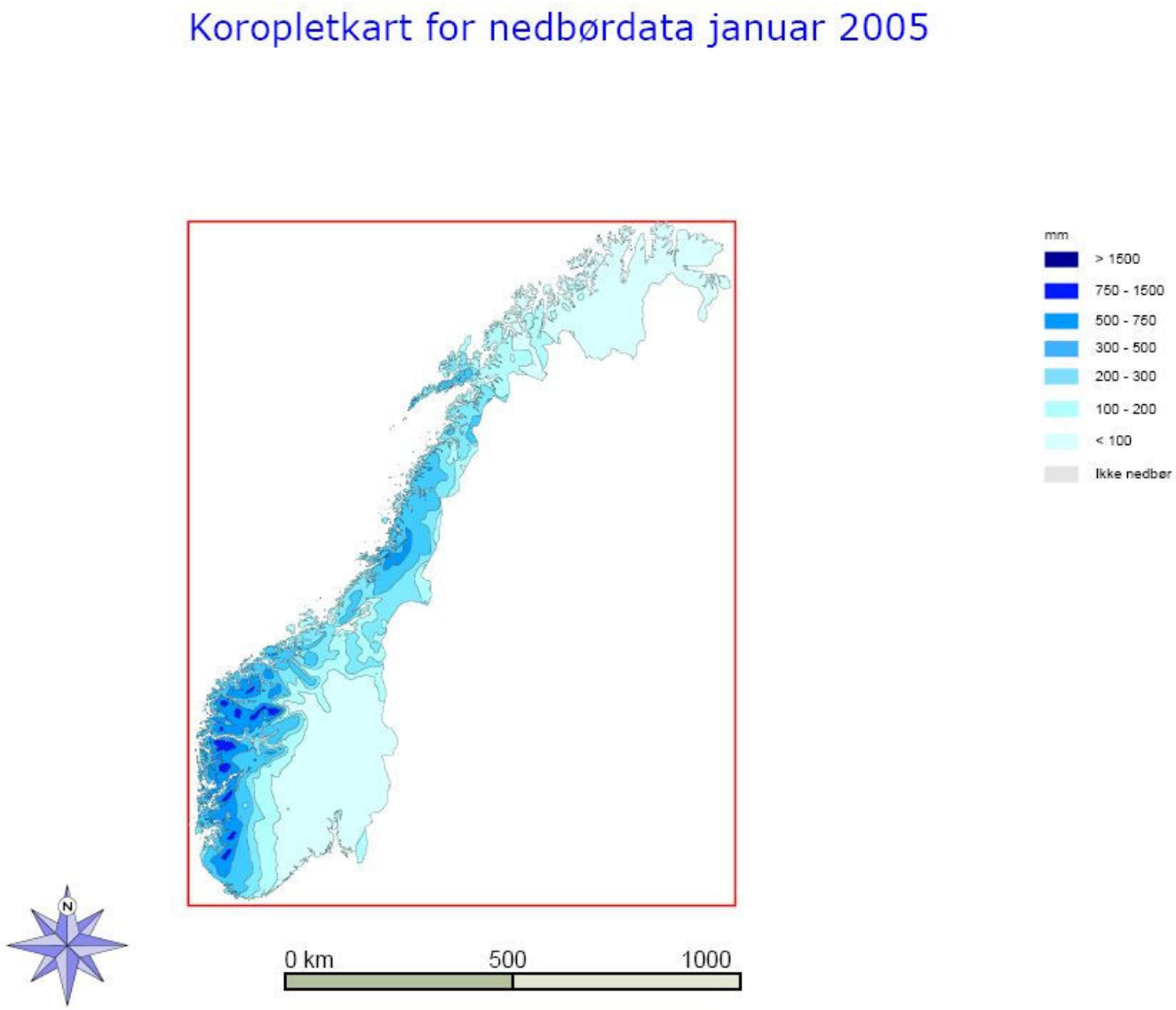
Figur 57: <glyph> eksempel

I et kartdokument bruker man tekst typisk til tegnforklaringer og for å beskrive eller lage titler på kartelementer. Tradisjonelt har man tilordnet en fargekode som representerer nedbørmengden, samt tall og måleenhetsbeskrivelse av hva fargekoden representerer. Kartet må ha titler – hvilket geografisk område er det vi ser på? – hvilken tidsperiode? I tillegg er det vanlig med en del hjelpetekster til interaktive verktøy, selv om intuitive verktøyknapper er gjengangere i mange applikasjoner (zoom og markeringsknapper). Vi har sett utforming av tekst kan gjøres på utallige måter, dette kan brukes til å lage attraktive visuelle løsninger med tekst.

Til nå har vi sett på et knippe grunnleggende og relevante SVG elementer. Vi har sett hvordan attributter til disse SVG elementene kan forme utseende og posisjon på grafikken. Jeg har vist litt om hvor allsidig SVG kan være når det gjelder grafisk utforming av vevsider, men også hvilke kontrollmuligheter man har over elementene. Dette er ikke alt – vi skal i neste avsnitt se på hvordan vi kan utøve kontroll og endringer på elementene, samt hvordan hente inn kartdata i et dokument fra noe annet enn fra et annet SVG dokument. Temaet er dynamikk og interaktivitet.

Jeg avslutter dette avsnittet med et relevant, men enkelt kartdokument som viser nedbørdata som et koropletkart. Se Figur 58.

Figur 58: eksempel på grafisk utforming i SVG



8.6 Interaktive og dynamiske kartdokument i SVG

Den mest basale metoden for å gi dynamiske egenskaper til et SVG-dokument er å bruke animasjon.
Animasjon kan deles inn i 5 karakteristiske typer.

- **Bevegelse og deformering**. Endre lokasjon, rotasjon, skalering og andre koordinat attributter i et kontinuerlig eller diskret tidsintervall.
- **Endring av egenskaper**. Eksempelvis å endre visse CSS egenskaper som opasitet til forskjellige verdier etc. i et tidsintervall.

- **Fargeanimasjon**. Endre fyllfarger, strekfarger og gradientfarger med mer.
- **Bildeanimasjon.** Endre på hvilket bilde som visualiseres over tid.
- **Interaktivitet.** Animasjon initiert av hendelser fra brukeren med datamus eller tastaturet.

Animasjonelement er et subelement som tilordnes en geometrisk element eller gruppe i SVG.

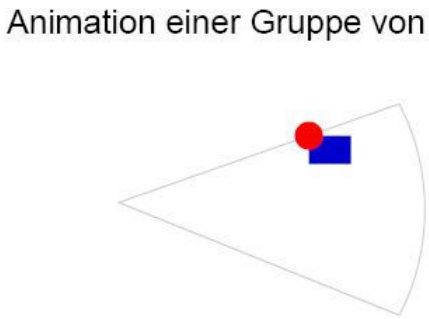
Jeg vil ikke drøfte animasjon inngående, fordi det primært er mer et attraktivt innslag i et kartdokument slik jeg ser det. Eksempelvis kunne man vise en animasjon hvor en sky skiftevis slipper regndråper og snøfnugg for ytterligere å understreke sammenhengen mellom kart og hva som vises.

Her følger en kodesnutt som illustrerer en type animasjon (eksemplet er hentet fra http://svglbc.datenverdrahten.de/?doc=g_animateMotion&znr=on):

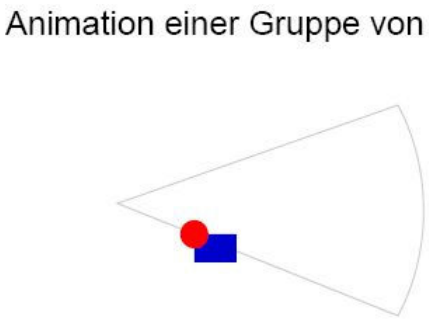
```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd" [
  <!ATTLIST svg xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink">
]>

<svg xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink">

  <title>SVG - Learning By Coding</title>
  <desc>SVG-Spezifikation in Beispielen</desc>

  <text x="20" y="30" style="fill: #000; font-size: 24px">
    Animation einer Gruppe von Objekten</text>

  <path d="M 100,150 L 300,80 A 100,130 0 0,1 300,230 Z"
    style="fill: none; stroke: #CCC; stroke-width: 1px"/>

  <g>
    <rect x="0" y="0" width="30" height="20" style="fill: #00C"/>
    <circle cx="0" cy="0" r="10" style="fill: #F00"/>
    <animateMotion begin="0s" dur="10s" repeatDur="indefinite"
      path="M 100,150 L 300,80 A 100,130 0 0,1 300,230 Z"/>
  </g>
</svg>
```

Eksemplet viser hvordan vi lager en animasjon i SVG, men den viser også hvordan animasjon kan grupperes til å gjelde flere elementer (her sirkel og rektangel) i synkron bevegelse. Se Figur 59 og Figur 60, som illustrerer samme SVG dokument ”samplet” på to forskjellige tidspunkter (t1, t2).

Figur 59: posisjon ved t1[http://svglbc.datenverdrahten.de/?doc=g_animateMotion&znr=on]



Figur 60:posisjon ved t2 [http://svglbc.datenverdrahten.de/?doc=g_animateMotion&znr=on]



Vi skal nå se på hvordan lage interaktive SVG kartdokumenter. Det er tre metoder for å lage interaktiv grafikk i SVG. Det er <a> elementet, SMIL (Synchronized Multimedia Integration Language) hendelser og skripting av hendelsesfunskjoner.

I et kartdokument kan man tenke seg å bruke hyperlenker for å lenke til eksterne ressurser som vedrører nedbørdata. Meteorologisk institutt har blant annet laget vevsider til et meteorologisk leksikon for termer og værfenomener innen meteorologi.
En annen måte å bruke hyperlenke på, er å lenke inn annen relevant kartinformasjon sammen med eksisterende kartinformasjon, som man normalt ikke får plass til sammen. Dette er en måte å la brukeren benytte lenker for å lenke inn informasjon og se dette i sammenheng med resten av kartdokumentet.

Et siste eksempel på bruk av lenker kunne også være å lage et grafisk lag bestående av prikker som representer målestasjoner. Prikkene med en brukervennlig størrelse for å kunne bruke datamuspekeren kan være lenker med målestasjonsnavn som kan lenkes til metadata om en målestasjon og annen relevant informasjon om målestasjonen.

<a> elementet fungerer på mange måter på samme måte som <a> merket i HTML/XHTML. Elementet fungerer som en lenke til andre ressurser som indikert med `xlink:href` attributtet. Alt som er kodet inne i dette elementet vil bli behandlet som en del av lenken. Unntaket er at et nøstet <a> element vil overstyre lenken.

Kodesnutt med bruk av <a> element:

```
.
.
 <defs>
    <style type="text/css"><![CDATA[
      text {font-family:verdana; font-size:16px;}
    ]]>
    </style>
  </defs>

 <a xlink:href="http://www.w3.org/"
     xlink:title="Lenke til www.w3.org"
     xlink:show="new">
    <rect x="10" y="10" ry="5" width="40" height="40"
    style="fill:chartreuse; stroke:black;" />
  </a>
  <text x="60" y="40">Lenke til www.w3.org</text>
</svg>
```

Figur 61: kodesnutt visualisert

Ved å klikke med musen på det grønne rektangelet blir man videresendt til W3C sine vevsider.

Interaktive SVG-dokumenter med bruk av SMIL er en annen måte å lage animasjon inn i et kartdokument. Hvor essensielt det et å bruke SMIL i et kartdokument kan diskuteres, men det kan gi visuelle attraktive løsninger, og derfor verdt å nevne i denne konteksten. Eksempel på animasjon i et kartdokument kan som nevnt i innledningen til dette avsnittet være en animasjon av en nedbørsky. I prinsippet kan man animere de flest elementer i SVG. Personlig er jeg tilhenger av en mer nøktern stil, og vil heller oppfordre til å bruke slike visuelle effekter med fornuft. For mye visuelle effekter kan overskygge hva man egentlig fokuserer på i et kartdokument og kan være irriterende hvis ikke hensiktsmessig utført.

Hvordan implementere SMIL i et kartdokument?

Nøkkelen til SMIL interaktivitet i SVG er å bruke **`begin`** og **`end`** attributtet for gjennomløp i et tidsintervall. Det er også mulig å starte og stoppe animasjonen basert på hendelser. I skrivende stund er slike hendelser i SVG basert på museaktivitet som ”mouseover”, mousemove”, ”mouseclick”, ”mouseup” og ”mousedown”. I tillegg gjelder aktiviteter som ”focusin”, ”focusout” og ”activate”. Den siste hendelsesstyringen er den mest vanlige.

Jeg vil lage en animert nedbørsky som feller ut regndråper.

Et relevant nedbørsky kodesnutteksempel:

```
.
<g id="nedbørsky">
      <g style="fill:none;stroke:#555555;fill-rule:nonzero;stroke-
dasharray:3.6;clip-rule:nonzero;stroke:#000000;stroke-miterlimit:4;stroke-
width:6;stroke-linecap:round;stroke-dasharray:6;">
      <path d="M3.6,21.9L34,52.3"/>
      <path d="M15.2,22.3l30.4,30.4"/>
      <path d="M29.2,20.3l30.4,30.4"/>
      <path d="M37.2,22.3l30.4,30.4"/>
      <path d="M46,18.7l30.4,30.4"/>
      <animate attributeName="stroke-width" attributeType=”CSS” from="0"
to="3" dur="2s" repeatCount="indefinite" accumulate="none"
additive="replace" calcMode="linear" fill="remove" restart="always"/>
      </g>
      <path style="fill:#494848;stroke:#000000;stroke-width:0.0;"
d="M73.2,18.9c-1.8-4-6.2-7-11.6-8C59.8,5.6,53.5,1.7,46,1.7 c-2.1,0-4.1,0.3-
6,0.9c-2.7-1.5-6-2.4-9.6-2.4c-5.8,0-10.9,2.3-13.7,5.9c-0.1,0-0.2,0-
0.3,0C7.4,6,0.1,11.6,0.1,18.5 c0,0.5,0,1.1,0.1,1.6c-0.1,0.2-0.1,0.5-
0.1,0.7v7.7c0,1.5,1.6,2.8,3.6,2.8h66.8c2,0,3.6-1.2,3.6-2.8v-7.7
C74.2,20.1,73.8,19.4,73.2,18.9z"/>
</g>
</svg>
```

Figur 62: eksempel på nedbørskyanimasjon

Interaktivitet utviklet med skripting og bruk av DOM er mer relevant i et kartdokument og helt essensielt for å utføre en rekke operasjoner både implisitt og eksplisitt av klient og bruker.
Vi har allerede drøftet operasjoner som zooming, panering, skalering og temporal datautvelgelse, samt en rekke andre operasjoner som er ment å tilordnes interaktive verktøy i et kartdokument.
I så måte er et kartdokument en relativt kompleks affære hvor en hendelse nødvendigvis må generere andre hendelser.
For å forklare det siste utsagnet, la oss gripe tak i følgende hendelse: zooming, enten inn eller ut zooming. Det som er nødt til å skje ved siden av at kartet skal forstørres eller forminskes, er at målestokken må også oppdateres. Har man i tillegg lagt inn koordinatreferanser som beskriver den geografiske utstrekningen som definert av et rektangel rundt selve kartobjektet, må også dette endres til riktige koordinater.
Et annet et eksempel vil være, at man henter inn nye nedbørdata, så vil man typisk måtte oppdatere all grafikk som avgir geografisk informasjon. Muligens må man og endre på nedbørforklaringene, for å fremheve fargekode og nedbørintervaller som er mer i samsvar med det som vises av nedbørdata i kartobjektet. Selvfølgelig må all tekstinformasjonen også oppdateres. Hva ser vi på? Hvilken tidsperiode?

Typisk vil mer informasjon i et kartdokument nødvendigvis øke kompleksiteten.

Ved transformasjoner av kartobjektet, som for eksempel rotasjon, vil man nødvendigvis også måtte dreie kartkompasset i riktig retning. Legger man inn en mengde visuelle effekter, så må de også gi korrekt informasjon hele tiden.

Et annet faktum (i skrivende stund) som taler for ”nøkternhet” er hvordan SVG-standarden er implementert hos klienten. De fleste nettlesere i dag som har støtte for SVG implementerer et subsett av standarden (Opera og delvis Mozilla er hederlig unntak). Bruk av programtillegget Adobe SVG Viewer gir heller ingen absolutte garantier for alle aspekter i SVG standarden er dekket. Programtillegget fungerer ikke feilfritt i alle nettlesere. Dette er dokumentert i http://www.adobe.com/svg/indepth/pdfs/CurrentSupport.pdf. Selv om man tilsynelatende gjør alt korrekt, så er det mulig at utilsiktede effekter oppstår i visualiseringen som følge av mer komplekse løsninger

Jeg nevnte at Viewer ikke fungerer 100 % likt i alle nettlesere, men hvis man leser dokumentasjonen for Viewer, så finner man fort ut at det absolutt er færre hensyn å ta enn å måtte skrive kode som omgår subsett implementasjon av SVG og Javaskript. I tillegg kan man hevde at å kun basere seg på støtte i nettlesere er ekskluderende for alle nettlesere som ikke har innbygget fortolkere av SVG og/eller ECMA skript. Nå kan man selvfølgelig argumentere mot dette, ved å hevde at de mest populære nettlesere i dag har delvis eller full støtte for standardene. Personlig synes jeg dette ikke er et godt argument. Utviklere av nettlesere må enten ha som målsetting å implementere standarden til fulle, eller støtte bruk av programtillegg. Jeg vil også hevde at mangelfull implementasjon hemmer bruken av SVG standarden på mange områder.

Mitt sluttpoeng er at fokus hele tiden må være på korrekt informasjon i det som visualiseres. En liten slurvefeil trekker helhetsinntrykket ned og det er fort gjort å begynne å tvile på hvor korrekt da resten av informasjonen er.

Vi skal nå se konkret på SVG-koding, skripting og bruk av DOM for å utvikle interaktive SVG dokumenter. Jeg vil også drøfte bruk av programtillegget Adobe SVG Viewer fordi jeg hevder at bruk av programtillegg er veien å gå ut i fra status på implementasjonen av SVG-standarden i skrivende stund. Et tilleggsmoment er at Adobe SVG Viewer har en innebygd ECMA-skriptmotor i tillegg til SVG fortolker. Dette faktumet gjør det enda mer attraktivt å benytte Viewer for å redusere kompleksiteten og sikre konformitet hos klienten og brukeren.

Skripting i SVG er mye likt det som gjøres i HTML/XHTML. Hendelsesstyrte funksjoner håndterer eksterne hendelser. Javaskript eller ECMA-skript (standard Javaskript) er støttet i de fleste SVG-versjonene. Dette er ikke et krav og man bør derfor teste implementasjonen (jamfør min diskusjon over).

For å beskrive bruk av skripting og DOM vil jeg ta utgangspunkt i enkle eksempler for å fremheve poenget med et minimum av kode. Eksemplene er hentet fra http://www.carto.net/papers/svg/manipulating_svg_with_dom_ecmascript/.

Eksempel 1:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
```

```
"http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" width="300" height="300">
  <script type="text/ecmascript">
    <![CDATA[
      function showRectColor() {

alert(document.getElementById("myBlueRect").getAttributeNS(null,"fill"));
      }

      function showRectArea(evt) {
        var width = parseFloat(evt.target.getAttributeNS(null,"width"));
        var height = parseFloat(evt.target.getAttributeNS(null,"height"));
        alert("The rectangle area is: " + (width * height));
      }

      function showRootChildrenNr() {
        alert("Nr of Children:
"+document.documentElement.childNodes.length);
      }
    ]]>
  </script>
  <g id="firstGroup">
    <rect id="myBlueRect" width="100" height="50" x="40" y="20" fill="blue"
onclick="showRectArea(evt)"/>
    <text x="40" y="100" onclick="showRectColor()">Click on this text to
show rectangle color.</text>
    <text x="40" y="130">Click on rectangle to show rectangle area.</text>
    <text x="40" y="160" onclick="showRootChildrenNr()">Click on this text
to show the number of child
     <tspan x="40" dy="20">elements of the root element.</tspan></text>
  </g>
</svg>
```

Eksemplet viser tre metoder for å lage en referanse til elementene.

1. `document.getElementById(unikId)`: Metoden lager en referanse til et element med en unik ID. Det er en metode i `document` object og brukes her i funksjonen `showRectColor()`.
2. `document.documentElement`: Metoden lager en referanse til rot eller dokument elementet. Det er også en metode i `document` object. Her er den brukt i `showRootChildrenNr()`. Samme resultat kan oppnås med å bruke egenskapen document.`rootElement`.
3. `evt.target`: Metoden lager en referanse ved hjelp av `evt` objektet. Hendelseobjektet er et spesielt objekt og genereres etter at mus, tastatur, dokument, DOM eller en status hendelse har skjedd. Den har flere egenskaper og metoder – blant dem er at den vet hvilket element som har generert hendelsen. Fordelen til metoden `evt.target` er at elementet som skaper hendelsen ikke trenger en unik ID og kan brukes av mange elementer samtidig. Her er metoden brukt i funksjonen `showRectArea()`.

Etter å ha laget referanse til et element, er det mulig å lese attributtene til dette elementet ved å bruke metoden *`element.getAttributeNS()`*. Funksjonen returnerer verdien til attributtet, denne verdien er alltid en tekststreng. For å returnere tallverdier brukes funksjonene *`parseFloat`* eller *`parseInt`* som vist i funksjonen showRectArea*`(evt)`*.

Selv om metoden *element.getAttribute()* vil fungere, så er det anbefalt å bruke funksjonen *element.getAttributeNS()* fordi den fungerer i dokumenter med mange navnerom. Metoden tar to argumenter: navnerom og attributtnavn. Hvis navnerommet er SVG, kan man bruke NULL som argument i stedet.for full navnerom-definisjon. Det er fordi metoden automatisk erstatter vertsnavnerommet med NULL.

Funksjonen *showRootChildrenNr()* returnerer antall barnenoder i rotelementet. Barnenoder er en rekkefølge (array) som inneholder referanse til alle barneelementer av et element. *childNodes.length* returnerer antallet i rekkefølgen.

Eksempel 2:

```
.
.
      <script type="text/ecmascript"><![CDATA[
            function showContentAndRelatives(evt) {
                  //get reference to text element
                  var textElement = document.getElementById("myText");
                  //get reference to parent of element
                  var parent = textElement.parentNode;
                  alert("the parent node has id
'"+parent.getAttributeNS(null,'id')+"'\nNodename is '"
                              +parent.nodeName+"'");
                  //get a reference to the first child of the element
"myText"
                  var child = textElement.firstChild;
                  //loop over all childs
                  while (child != null) {
                        //see if child is a tspan and has child nodes
                        if (child.nodeName == "tspan" &&
child.hasChildNodes()) {
                              //see if firstChild is of nodeType "text"
                              if (child.firstChild.nodeType == 3) {
                                    alert("child's text
content="+child.firstChild.nodeValue);
                              }
                        }
                        child = child.nextSibling;
                  }
                  alert("the content of the second tspan Child is:
"+textElement.childNodes.item(1).firstChild.nodeValue);
            }
      ]]></script>
      <g id="firstGroup">
            <text id="myText" x="50" y="30"
onclick="showContentAndRelatives(evt)">
                  <tspan>Click on text</tspan>
                  <tspan x="50" dy="15">to get parent node data</tspan>
                  <tspan x="50" dy="15">and to see the text node values
</tspan>
                  <tspan x="50" dy="15">of each line</tspan>
                  </text>
      </g>
</svg>
```

Dette eksemplet er ment å vise hvordan man lager en referanse til foreldre, barne- og søsken noder.

Egenskapen *element.parentNode* lager en referanse til foreldrenoden i dette eksemplet. ID og nodenavn til foreldrenoden "varsles" for å vise at det er en aksess til foreldrenoden.

Bruk av *element.firstChild* egenskapen lager en referanse til den første barnenoden til et element. Ved å bruke denne referansen traverserer man gjennom alle barnenoder til <text> elementet inntil NULL verdi er returnert. På slutten av sløyfen returnerer egenskapen *child.nextSibling* returnerer en referanse til søskennoden av barneelementet. I sløyfen testes det for om *child.nodeName* egenskapen returnerer "tspan" og om barnenoden også har barnenoder. Dette er nødvendig siden det er en teoretisk mulighet for at så er tilfelle. Spesielt også fordi ikke visuelle tegn som ny linje eller mellomrom mellom elementer vil få en referanse.
I den samme testen, testes det for om barnenoden er av *nodeType* "3" som er en tekstnode i dette tilfellet <tspan/> elementet

Tabell over nodetyper:

1	ELEMENT_NODE
2	ATTRIBUTE_NODE
3	TEXT_NODE
4	CDATA_SECTION_NODE
5	ENTITY_REFERENCE_NODE
6	ENTITY_NODE
7	PROCESSING_INSTRUCTION_NODE
8	COMMENT_NODE
9	DOCUMENT_NODE
10	DOCUMENT_TYPE_NODE
11	DOCUMENT_FRAGMENT_NODE
12	NOTATION_NODE

Ovenstående eksempler er inkludert fordi de gir elementære kunnskaper i hvordan traversere SVG dokumenter og få tak i elementer, attributter og deres verdier. Typiske oppgaver vil være å endre attributtverdier, samt lage nye elementer eller verktøy.

Interaktive verktøy kan utvikles med ECMA-skript eller SVG-elementer og hendelser. ECMA-skript må nødvendigvis ikke skrives inn sammen med resten av dokumentet, men kan lagres i egne filer og lenkes inn.

Følgende kodesnutt er eksempel på ekstern ECMA-skript deklarasjon i SVG.

```
<script type="text/ecmascript"
xlink:href="../../resources/helper_functions.js" />
```

Se også http://www.w3.org/TR/2000/WD-SVG-20000629/ecmascript-binding.html for en komplett referanse av ECMA-skript binding i SVG DOM.

Eksempler på grafikk for interaktive verktøy:

Figur 63: panering

Figur 64: skriv ut

Figur 65: zoom inn område

Figur 66: zoom inn

Figur 67:zoom ut

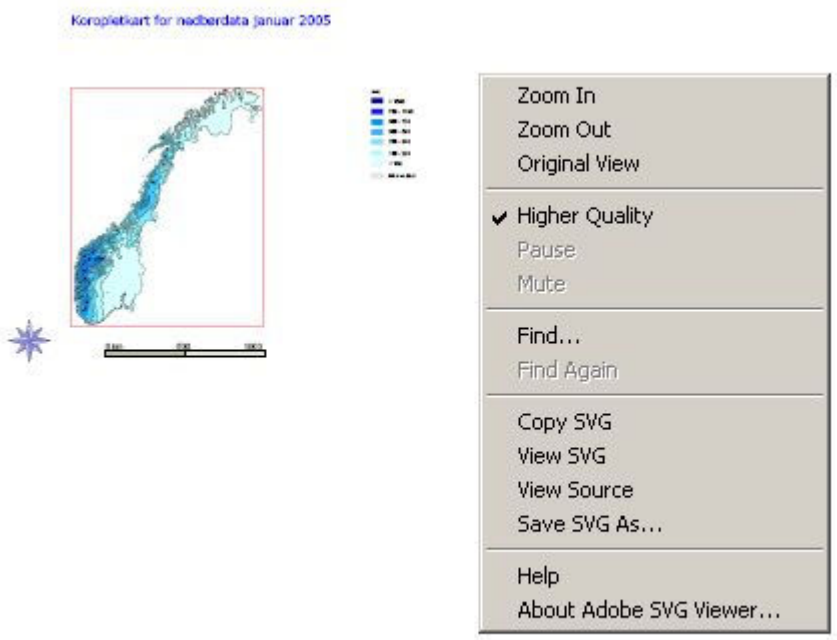
Disse grafiske elementene kan enten utvikles i SVG, punktgrafikk og/eller Javaskript. SVG-standarden definerer elementer for punktgrafikk. Se http://www.w3.org/TR/SVG/struct.html#ImageElement for detaljert informasjon om elementet for punktgrafikk.

Til nå har vi implisitt brukt innebygd støtte for ECMA-skript i eksemplene.

Hvordan kode i SVG slik at jeg eksplisitt kan benytte Adobe SVG-Viewer SVG-fortolker og ECMA-skript motor?
Det har hele tiden vært min strategi for å unngå å lage alle mulige unna manøvrer for å styre unna problemer med nettlesere.

Når man har installert Adobe SVG Viewer (ASV), vil man oppdage at kontekstmenyen (høyreklikkmeny) i nettleseren har fått oppført interaktive verktøy.

Figur 68: AVS kontekstmeny (høyreklikk)



Av Figur 68 kan man se at jeg har benyttet meg av menyen og zoomet kartet nesten ut av fokus. Et kjekt verktøy før man selv kommer i gang med å lage interaktive verktøy.

Man kan også i SVG koden foran legge inn prosesseringsinstruksjoner til ASV:

`<?AdobeSVGViewer save="snapshot"?>` før <svg> elementet betyr av ASV kan lagre nåværende DOM status i stedet for dem som ble satt inn under lasting av dokumentet.

Den generelle syntaksen er:

```
<?AdobeSVGViewer [resolution="number"] [save="no|snapshot|original"]?>
```

Siden versjon 3.0 har Adobe lagt til en signifikant egenskap i ASV nemlig Javaskript (JS) motor. Før denne versjonen kom ut baserte man seg på JS støtten i nettleseren. Bruk av JS motoren i ASV gir en rekke fordeler. For det første er motoren konform med ECMA-spesifikasjonen, og Adobe har brukt Mozilla's SeaMonkey motor, noe som i utgangspunktet er forbundet med pålitelighet. Konsekvensen av å bruke den innebygde motoren i ASV er at skriptene vil teoretisk kjøre like godt i Internet Explorer (IE), Netscape, Opera eller Mozilla uavhengig av om nettleserne er installert på forskjellige plattformer som Mac, Windows (CE, XP) eller Linux. Jeg sier teoretisk fordi dette er en sannhet med modifikasjoner. ASV har en del uoverensstemmelser med nettleserne IE og Netscape, som jeg tidligere nevnte.

Å aktivisere ASV JS motor er relativt enkelt. Motoren må aktiviseres eksplisitt, den er ikke aktivisert selv om vi benytter oss av SVG fortolkeren. Aktivisering utføres med å bruke et attributt for Adobes navnerom.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
```

```
 "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">

<svg  xmlns="http://www.w3.org/2000/svg"
      xmlns:a3="http://ns.adobe.com/AdobeSVGViewerExtensions/3.0/"
      a3:scriptImplementation="Adobe">

  <script a3:scriptImplementation="Adobe" type="text/ecmascript">
    <![CDATA[

      // EcmaScript code

    ]]>
  </script>

  <!-- SVG contents -->
</svg>
```

Fet skrift fremhever den nødvendige koden.

Skripting utføres på vanlig måte – her et relevant eksempel for zooming i SVG.

Skript:

```
var zoom_viewbox = false
function actualise_anim(evt)
{
      node = evt.target.ownerDocument.getElementById("root")
      if (zoom_viewbox == false)
      {
            zoom_viewbox = true
            xm = -2690 + evt.clientX * 5375 / 700
            ym = -100 + evt.clientY * 4421 / 576
            xsvg = Math.round( -2690 + xm * 5375 / 700 )
            ysvg = Math.round( -100 + ym * 5375 / 700 )
            node.setAttributeNS(null, "viewBox", (xsvg-644).toString() + "
"+ (ysvg-531).toString() + " 1288 1061")
      }
      else
      {
            zoom_viewbox = false
            node.setAttributeNS(null, "viewBox" , "-2690 -100 5375 4421")
      }
}
```

SVG kode for å fange hendelsen ”museklikk”:

```
<-- We play with viewBox attribute of svg -->
<svg id="root" width="700" height="576" viewBox="-2690 -100 5375 4421"><g
onclick="actualise_anim(evt)">
<-- path elements for countries -->
</g>
</svg>
```

Ytterligere muligheter foreligger med programtillegget ASV aktivisert. Vi kan lage brukerspesifiserte kontekstmenyer med bruk av høyre museklikk.

Her vises standard kontekstmeny i XML:

```
  <defs>
     <menu id='NewMenu' xmlns='http://foo' onload='GetPosition( evt )'>
        <header>Custom Menu</header>
        <item action='ZoomIn'>Zoom &In</item>
        <item action='ZoomOut'>Zoom &Out</item>
        <item action='OriginalView'>&Original View</item>
        <separator />
        <item onactivate='MyFunction()'>&My Function</item>
        <menu>
        <header>Submenu</header>
        <item onactivate="alert('Item1')">Item1</item>
        <item onactivate="alert('Item2')">Item2</item>
        <item onactivate="alert('Item3')">Item3</item>
        </menu>
        <separator />
        <item xmlns='http://www.w3.org/1999/xlink'
xlink:href='inserted_link.html'
          target='resource window'>Menu Link...</item>
        <separator />
        <item action='Quality'>Higher &Quality</item>
        <item action='Pause'>&Pause</item>
        <item action='Mute'>&Mute</item>
        <separator />
        <item action='Find'>&Find...</item>
        <item action='FindAgain'>Find &Again</item>
        <separator />
        <item action='CopySVG'>&Copy SVG</item>
        <item action='ViewSVG'>&View SVG...</item>
        <item action='ViewSource'>View Sourc&e...</item>
        <item action='SaveAs'>&Save SVG As...</item>
        <item action='SaveSnapshotAs'>Sa&ve Current State...</item>
        <separator />
        <item action='Help'>&Help...</item>
        <item action='About'>About SVG Viewer...</item>
     </menu>
  </defs>
```

Den kan også deaktiveres. Kontekstmenyen i ASV er egentlig en DOM instans og er tilgjengelig via tilgangsfeltet *`contextMenu`* i *`window`* objektet.

I et temporalt og interaktivt kartdokument er datautvelgelse en viktig operasjon. Jeg vil drøfte mulighetene for å gjøre dette i SVG. Å hente data er et viktig aspekt i enhver kartapplikasjon, og vi skal se at det overraskende nok ikke er støtte for dette direkte i SVG versjon 1.1.

Problemstillingen er å koble til en tjener og hente ut nedbørdata på SVG format inne i et SVG kartdokument.
Flash-spilleren har her et fortrinn sammenliknet med SVG. Flash-spilleren kan lage en XML-tilkobling for å laste ned data. Verken SVG eller DOM-standarden spesifiserer metoder for nettverkstilkobling for nedlasting og lagring. DOM 3 standarden, som foreløpig er et utkast, tilbyr en slik metode. Adobe har sett behovet i forkant og tilbyr SVG-utviklere metoder for å koble til en tjener for nedlasting og lagring av data pluss litt ekstra via programtillegget.

Enda et argument for å vurdere programtillegget ASV?

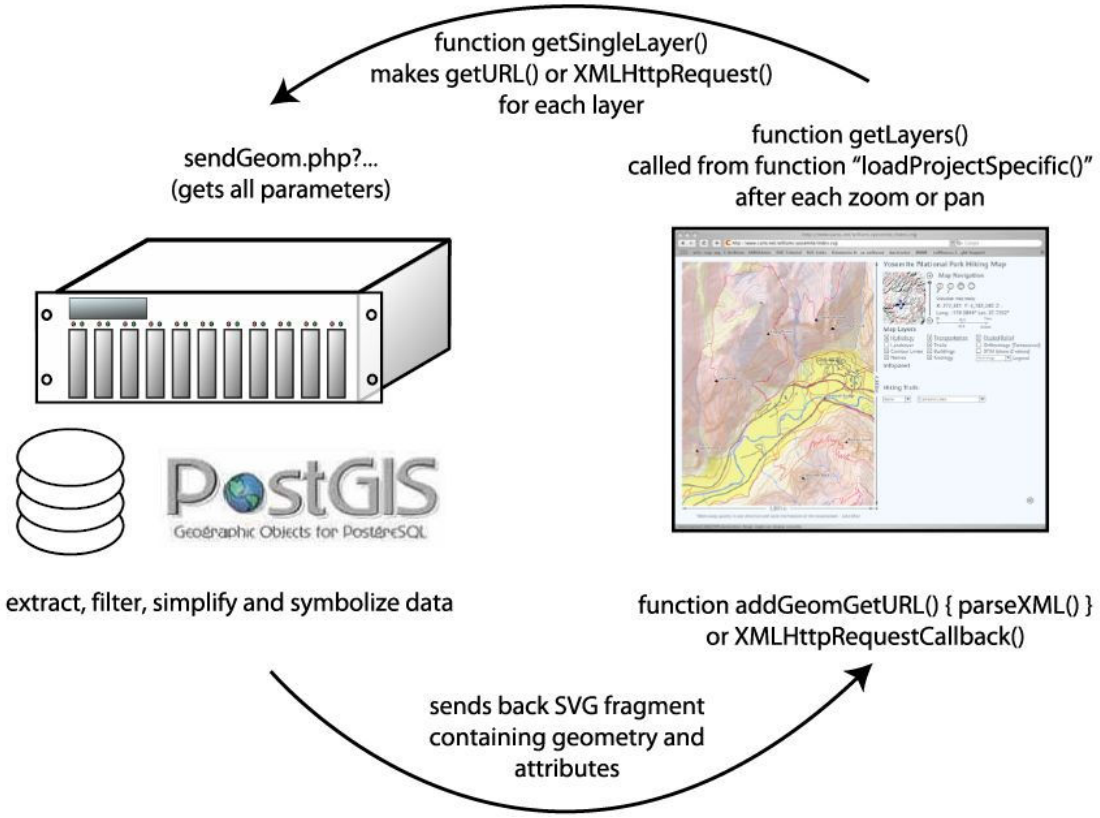
Metodene som er implementert er `getURL()` og `postURL()`.
Begge er definert i `window` objektet og tilbyr funksjonalitet for nedlasting og opplasting av data til en gitt URL (Uniform Resource Locator). Begge metodene krever en tilbakekall-metode (eng. callback) som parameter. Denne metoden blir automatisk overført til `AsyncURLStatus` objekt som gir noe informasjon om status på operasjonen og data som man mottar. Eksempel (se, http://www.xml.com/pub/a/2002/07/03/adobesvg.html):

```
function loadFile (fileName) {
      getURL(fileName, fileLoaded);
}

function fileLoaded (data) {
      var msg = '';
      if(data.success) {
            msg = data.content;
      } else {
            msg = 'Loading has failed';
      }
      alert(msg);
}
```

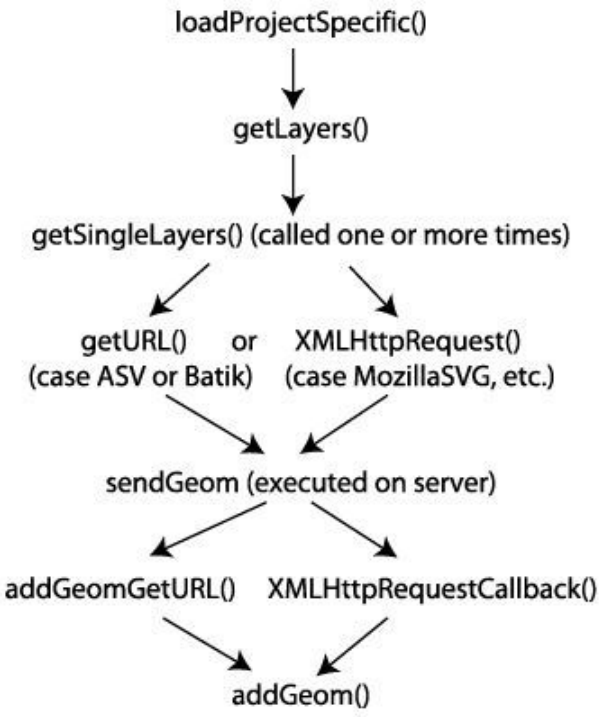
Jeg vil avslutningsvis vise metodikken for å hente nedbørdata fra en PostgreSQL database inn i et SVG kartdokument. Vi skal studere et kodeutvalg fra en kartapplikasjon. Vi skal kun se på bestemte aspekter ved applikasjonen, nærmere bestemt det å laste ned kartdata. Stikkord er PHP, AJAX, Javaskript, SQL samt prosessering på tjenersiden.

Eksempelet har jeg hentet fra Carto:net som jeg vil anbefale som en ypperlig kilde til informasjon med eksempler på bruk av SVG i kartapplikasjoner.
Se http://www.carto.net/papers/svg/postgis_geturl_xmlhttprequest/ for detaljert informasjon og eksempler. Jeg har benyttet meg av deres eksempel og illustrasjoner i dette for å drøfte metoder for datautvelgelse.

Figur 69: nedlasting av data i tilknytning til SVG



Figur 70: arbeidsflytskjema for applikasjonen



Jeg vil innlede med å forklare hva figurene illustrerer. Figur 69 illustrerer rammeverket i applikasjonen. Vi har en klient og nettleser som foretar en datautvelgelse mot en PostgreSQL database med PostGIS. Det går ikke frem av skissen at PostgreSQL er installert på en ekstern tjenermaskin, og det er heller ingen krav at så må være tilfelle. For diskusjonens skyld sier vi at PostgreSQL er installert på en ekstern tjener, fordi dette er en vanlig setting.

Hvis vi går i retning mot klokka, forteller figuren oss at vi har funksjoner som kjører på klientsiden (Javaskript) og som kaller en CGI applikasjon skrevet i PHP som kjører en spørring mot PostgreSQL. Spørringen resulter i at data returneres til klienten på SVG format. SVG-data håndteres av ytterligere flere funksjoner i nettleseren (vi forutsetter at nettleseren har en innebygd Javaskript fortolker). Funksjonene fortolker SVG-data, blant annet for visualisering i nettleseren.

Figur 70 detaljerer arbeidsflyten i det som jeg beskrev ovenfor med fokus på hvilke programmer og funksjoner som involvert, hvor alt kulminerer i funksjonen `addgeom()`.

Vi skal se nærmere på det som skjer uten å grave oss ned i detaljer i kodingen. All kode kan lastes ned og det er mulig å installere og kjøre applikasjonen. Dette har jeg allerede testet.

Forfatterne går meget detaljert til verks og vi hopper over innledningen med å lage en database og installere PostGIS i denne databasen samt eksportering av ESRI shapefiler til databasen Det er kartdata som skal visualiseres, og interessant nok dreier det seg om hydrologiske data.

Vi starter med CGI applikasjonen kodet i PHP for spørring mot databasen. Kodingen starter elementært med å definere tilkoblingsparametere mot databasen, altså brukernavn, tjenernavn etc.
Til CGI applikasjon skjer det en parameteroverføring som håndteres i koden, her kalt URL parametere.

Kallet til CGI applikasjonen:

```
sendGeom.php?layername=hydrology&xmin=244000&ymin=4150000&xmax=308000&ymax=
4231000&timestamp=123
```

Lagring av overførte parametere:

```
//get URL parameters
$layername = $_GET['layername'];
$myId = $layername.'_tempGeometry';
$xmin = intval($_GET['xmin']);
.
.
```

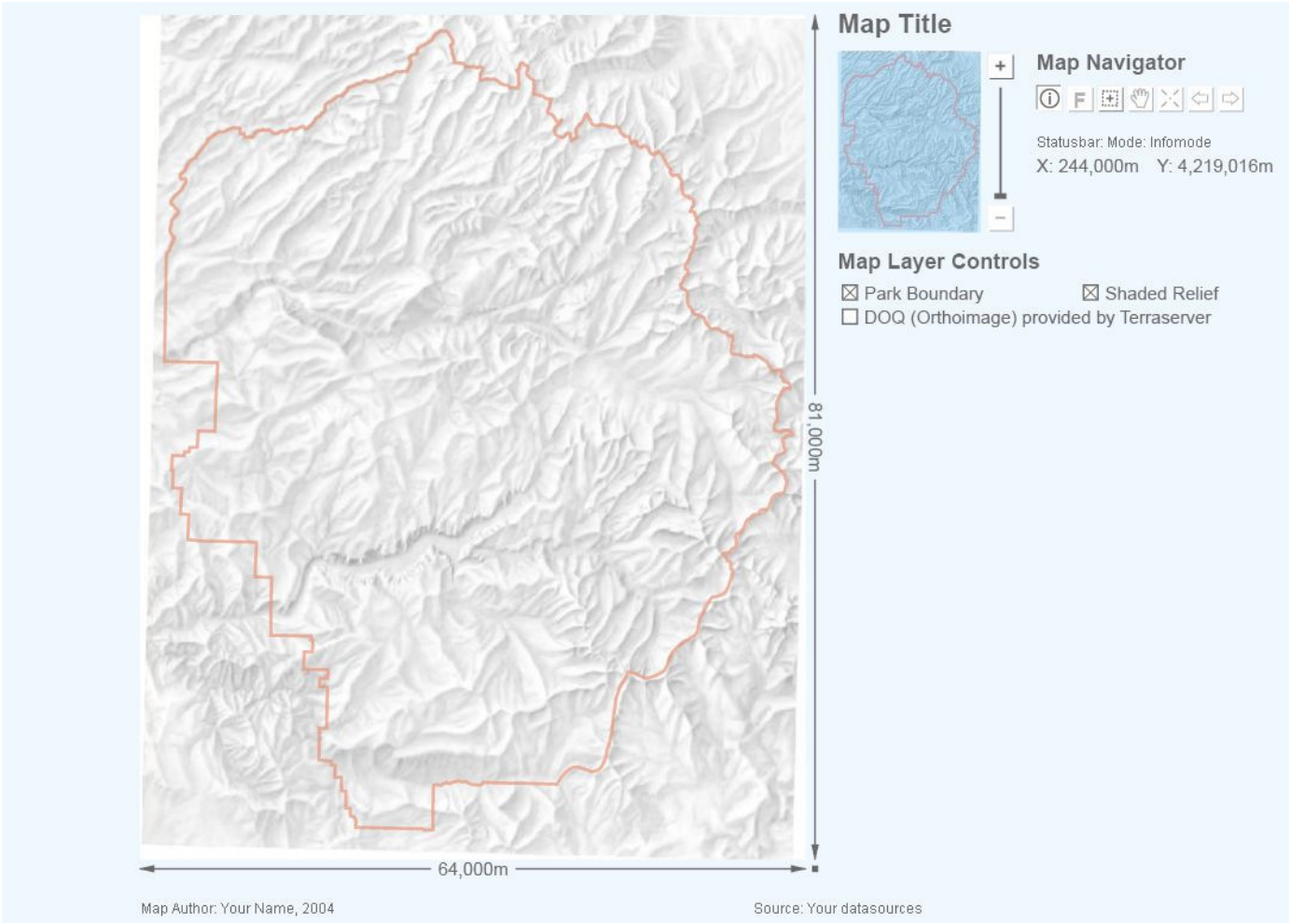
Videre i applikasjonen settes SQL strengen opp og tilkobling mot databasen og spørringen kjøres. Data returneres og formateres til et SVG dokument. Dette skjer egentlig i to steg. Selve spørringen får PostgreSQL til å formatere kartdata som <path> elementer med bruk av AsSVG() funksjonen i PostGIS. sendGeom.php applikasjonen grupperer data med <g> elementet. Det er viktig her å merke seg at det ikke er et komplett SVG dokument som returneres, men et SVG dokumentfragment.
Vi går ikke mer inn på detaljene her. Det vi kan si, er at CGI applikasjonen kunne like gjerne vært kodet i Perl eller i Java med bruk av servlets teknologien.

Selve kartdokumentet er kalt index.svg, og jeg viser kun innledende kode til dette dokumentet:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd" [
<!ATTLIST svg
      xmlns:attrib CDATA #IMPLIED
      xmlns:batik CDATA #IMPLIED
.
.

]>
<?AdobeSVGViewer save="snapshot"?>
<svg width="100%" height="100%" viewBox="0 0 1024 768"
xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:attrib="http://www.carto.net/attrib"
xmlns:batik="http://xml.apache.org/batik/ext" onload="init(evt);"
zoomAndPan="disable">
      <title>Loading WMS raster data from UMN Mapserver to an SVG mapping
application</title>
      <script type="text/ecmascript"
xlink:href="../resources/helper_functions.js"/>
      <script type="text/ecmascript" xlink:href="../resources/mapApp.js"/>
      <script type="text/ecmascript" xlink:href="../resources/timer.js"/>
      <script type="text/ecmascript" xlink:href="../gui/slider/slider.js"/>
      <script type="text/ecmascript" xlink:href="../gui/button/button.js"/>
      <script type="text/ecmascript"
xlink:href="../navigationTools/navigation.js"/>
      <script type="text/ecmascript"
xlink:href="../gui/checkbox_and_radiobutton/checkbox_and_radiobutton.js"/>
      <script type="text/ecmascript"><![CDATA[
            var myMapApp = new mapApp(false,undefined);
            var myMainMap;
```

Det som er verdt å merke seg her, er prosessinstruksjonen til ASV som er programtillegget i nettleseren. Videre er det en rekke eksterne filer med Javaskript som lenkes inn. En annen ting å merke seg, er attributtet i <svg> *zoomAndPan="disable".* Tilordning av verdien ”disable” til attributtet deaktiviserer zooming og panering i ASV (jamfør diskusjonen om kontekstmeny i ASV).
Betyr dette at zooming og panering ikke er mulig?
Svaret er nei. Det er kun kontekstmenyen i ASV for zooming og panering som er dekativisert. Zooming og panering utføres med hendelsesfunksjonene skrevet i Javaskript tilordnet interaktive verktøyknapper i kartdokumentet. Hvis zooming var aktivisert i ASV, så ville det få den uønskede effekten at hele SVG dokumentet ble forstørret eller forminsket i motsetning til SVG-fragmentet som utgjør kartobjektet. Se Figur 71.

Figur 71: Carto:net kartdokument eksempel



Vi skal studere funksjonene som initierer kallet til CGI-applikasjonen og funksjonene som håndterer mottak av data. De andre funksjonene er hendelsesstyrte funksjoner tilordnet interaktiv verktøygrafikk for blant annet zooming, panering og oppdatering av koordinater og annen kartinformasjon i henhold til endringer i kartdokumentet.

Funksjonen getURL() eller XMLHttpRequest kalles inne i funksjonen getSingleLayer(). Den sjekker om funksjonene er tilgjengelige for å kalles og utfører så et kall til en av dem eller gir en melding om at funksjonene ikke er tilgjengelig i nettleseren.
Funksjonen getURL() er implementert i ASV. XMLHttpRequest er egentlig et Microsoft ActiveX programtillegg som er implementert i mange nettlesere. Ingen av funksjonene er standarder.
Det som er interessant, er at de kan utføre asynkrone fjernkall. Med asynkron menes at de utfører fjernkallet, men stopper ikke kjøringen i påvente av en respons.
XMLHttpRequest skiller seg fra getURL() ved at den kan også utføre synkrone fjernkall samt har en rekke andre metoder og statussjekking. Se definisjonen av getSingleLayer() i koden som følger.

```
function getSingleLayer(layerId) {
      //build a url string
      var myUrlString =
"sendGeom.php?layername="+layerId+"&xmin="+myMainMap.curxOrig+"&ymin="+((my
MainMap.curyOrig + myMainMap.curHeight)* -
1)+"&xmax="+(myMainMap.curxOrig+myMainMap.curWidth)+"&ymax="+(myMainMap.cur
yOrig * -1)+"&timestamp="+myMainMap.timestamp;
      //call getURL() if available
      if (window.getURL) {
            getURL(myUrlString,addGeomGetURL);
      }
      //call XMLHttpRequest() if available
      else if (window.XMLHttpRequest) {
            //this nested function is used to make XMLHttpRequest
threadsafe
            //(subsequent calls would not override the state of the request
and can use the variable/object context of the parent function)
            //this idea is borrowed from
http://www.xml.com/cs/user/view/cs_msg/2815 (brockweaver)
            function XMLHttpRequestCallback() {
                  if (xmlRequest.readyState == 4) {
                        if (xmlRequest.status == 200) {
                                    //parse and import the SVG fragment
                                    var importedNode =
document.importNode(xmlRequest.responseXML.documentElement,true);
                                    //call function addGeom
                                    addGeom(importedNode);
                        }
                  }
            }
            var xmlRequest = null;
            xmlRequest = new XMLHttpRequest();
            xmlRequest.open("GET",myUrlString,true);
            xmlRequest.onreadystatechange = XMLHttpRequestCallback;
            xmlRequest.send(null);
      }
      //write an error message if either method is not available
      else {
            alert("your browser/svg viewer neither supports window.getURL
nor window.XMLHttpRequest!");
      }
}
```

For å motta data benyttes tilbakekallingsfunksjoner.
Tilbakekallingsfunksjonen *addGeomGetURL*() er implementert i ASV. Funksjonen sjekker om dataobjektet som er returnert fra tjener inneholder egenskapen *data.success*. Hvis den ikke gjenkjennes, har noe gått galt. Hvis alt går bra, blir XML-data fra *data.content* egenskapen tolket av funksjonen *parseXML*() (også en ASV implementering). Endelig blir funksjonen *addGeom*() kalt med det første "barnet" *DOCUMENT_FRAGMENT_NODE* som er mottatt fra funksjonen *parseXML*() som parameter.

For *XMLHttpRequest* er denne funksjonen kalt XMLHttpRequestCallback() styrt av hendelsen *onreadystatechange*. Funksjonen kalles når *readyState* endres. I dette eksemplet er vi kun interessert i *readyState* 4, som initieres når transaksjonen er sluttført. Vi bruker da egenskapen *responseXML* som inneholder XML-datafragmentet. Videre importeres *documentElement* fra fragmentet med kall til *importNode*() som igjen overfører det til

funksjonen *addGeom*() for videre behandling. I denne sammenhengen er det viktig at skriptet på tjeneren må sende korrekt `content-type header`: (se andre linje i sendGeom.php `header("Content-type: text/xml")`), ellers fungerer ikke egenskapen *responseXML*.

Tilbakekallingsfunksjon i ASV.

```
function addGeomGetURL(data) {
      //check if data has a success property
      if (data.success) {
            //parse content of the XML format to the variable "node"
            var node = parseXML(data.content, document);
            addGeom(node.firstChild);
      }
      else {
            alert("something went wrong with dynamic loading of
geometry!");
      }
}
```

`XMLHttpRequestCallback()` kalles i `getSingleLayer()` (se tidliger gjengitt kodesnutt for funksjonen).

```
//add the geometry to the application
//note that empty groups in the main-map have to exist
function addGeom(node) {
      //extract id of the layer
      var curDynLayer =
node.getAttributeNS(null,"id").replace(/_tempGeometry/,"");
      //extract timestamp
      var timestamp = parseInt(node.getAttributeNS(attribNS,"timestamp"));
      //compare current timestamp with timestamp of the node data
      if (timestamp == myMainMap.timestamp) {
            var myGeometryToAdd = document.getElementById(curDynLayer);
            //remove previous content
            if (myMainMap.dynamicLayers[curDynLayer].loaded == "yes") {
                  var tempGeometry =
document.getElementById(curDynLayer+"_tempGeometry");
                  myGeometryToAdd.removeChild(tempGeometry);
            }
            //append new content
            myGeometryToAdd.appendChild(node);
            //set loaded flag to "yes"
            myMainMap.dynamicLayers[curDynLayer].loaded = "yes";
      }
      //decrement layers to load
      myMainMap.nrLayerToLoad[myMainMap.timestamp.toString()]--;
      //set loadData text to hidden after last layer was loaded
      if (myMainMap.nrLayerToLoad[myMainMap.timestamp.toString()] == 0) {

      document.getElementById("loadingData").setAttributeNS(null,"visibilit
y","hidden");
      }
}
```

Dette eksemplet er viktig av to grunner. For det første viser det en metode for å hente kartdata på SVG-format via et CGI-skript. For det andre er metodikken som er benyttet den samme som benyttes i AJAX-metodikken.

8.7 Visualisering i 3D

Så langt i oppgaven har vi sett på 2D visualisering. En naturlig utvikling for å lage interessante visuelle vinklinger på nedbørdata og andre klimadata som kartobjekter er 3D-visualisering. For å fortsette diskusjonen med nedbørdata, så er 2D fremstillingen basert på x og y koordinater gitt av geografiske koordinater i et projisert plan. Stedfestet nedbørmengde har blitt visualisert som et koropletkart, TIN og/eller høydekurvekart.
For å visualisere nedbørdata i 3D (2.5D), så må attributtet nedbørmengde tilordnes z koordinaten i et tredimensjonalt koordinatsystem.
3D visualisering av nedbørdata kan være statisk eller dynamisk. Statisk vil si at nedbørdata er visualisert kun med en synsvinkel. Dynamisk innebærer at 3D visualiseringsverktøyet har innebygd kontroll for å kunne bevege og dreie på 3D objektet.
Hensikten med 3D visualisering av nedbørdata kan være å se nedbørdata i sammenheng med topografien.
I 2D ser vi hvor nedbøren faller geografisk. En 3D visualisering kan gi ytterligere informasjon om hvorfor nedbørmengden varierer fra sted til sted sett i et kartobjekt med topografisk dybdevisualisering. I tillegg kan animasjon benyttes til å legge til en temporal dimensjon.

Hvordan visualisere nedbørdata i 3D?

Det er mange måter å visualisere nedbørdata i 3D på. En måte er å kombinere topografisk informasjon med fargekoding av arealet: et koropletkart i 3D. Z koordinaten representerer da kun terrenghøyden og nedbørmengden et attributt til høyden. Motsatt – terrenget i et plan og z koordinaten er tilordnet nedbørmengde. Nedbørmengden kan være representert som ”stolper” eller kontinuerlige felter. En dreining av kartobjektet horisontalt og sidelengs vil da visuelt se ut som et stolpe- eller kurvediagram i et todimensjonalt kartesisk system. En slik horisontal dreining gir oss muligheten til å se 2D i alle mulige himmelretninger.
Med en perspektivisk eller aksonometriske projeksjon vil visualiseringen være et topografisk nedbørkart.
Med animasjon kan vi gi et temporalt aspekt av nevnte projeksjoner.

Hvilken teknologi skal man benytte til 3D visualisering?

Et naturlig valg er W3C standarden X3D som er etterfølgeren til VRML. X3D er et XML-basert markeringsspråk og er således en naturlig videreutvikling med utgangspunkt i 2D standarden SVG. I tillegg til XML-koding tilbyr X3D standarden SAI (Scene Authoring Interface) for at både vevapplikasjoner og ikke vevbaserte applikasjoner skal kunne bygge inn 3D data som opererer i sanntid. Se http://www.web3d.org/x3d/specifications/ for å få mer innsikt i X3D-spesifikasjonen.

X3D og SVG er de viktigste grafiske standardene og vil spille en viktig rolle i tiden som kommer. X3D er en standard i utvikling for diverse interaktive vevbaserte 3D løsninger, som kan integreres med multimedia på uavhengig av maskinvare, og er neste generasjon ISO (International Standards Organization) standard.

X3D standarden er organisert som et sett med komponenter hvor hver komponent utgjør et sett sammenkoblede funksjonaliteter bestående av forskjellige 3D objekter og/eller tjenester. X3D er altså modulært oppbygd for å kunne tilby tjenester på forskjellig nivåer for å gi utviklere muligheten til å implementere 3D grafikk uavhengig av maskinvareplattform.

Det overordnede nivået til komponenter er profiler. En profil er en sett med navngitte funksjonaliteter og krav som må oppfylles for at profilen skal bli konform med X3D-spesifikasjonen. Profiler utgjør funksjonelle lag som man kan velge for å få mest mulig effektiv utnyttelse av maskinvaren. X3D består i skrivende stund av 6 profiler som støttes av standarden, rangert fra ”Interchange” profilen som gir et minimum av funksjonalitet, til ”Immersive” profilen som støtter mer kompleks funksjonalitet som virtuelle verdener med navigering og sensorkontroll av miljøet.. Samtidig skal det være bakoverkompatibelt med VRML97. Andre tilgjengelige valgmuligheter gir mellomliggende grad av detaljering som f. eks interaktiv multimedia (MPEG-4 for eksempel), CAD3D (Computer Aided Design).

Her er et enkelt X3D-dokument med fokus på dokumentformalia:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.1//EN"
"http://www.web3d.org/specifications/x3d-3.1.dtd">
<X3D profile="Immersive" version="3.0"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
xsd:noNamespaceSchemaLocation="http://www.web3d.org/specifications/x3d-
3.0.xsd">
  <head>
    <meta content="HelloWorld.x3d" name="title"/>
    <meta content="Simple X3D example" name="description"/>
    <meta content="30 October 2000" name="created"/>
    <meta content="18 December 2006" name="modified"/>
    <meta content="Don Brutzman" name="creator"/>
    <meta

content="http://www.web3d.org/x3d/content/examples/Basic/HelloWorld.x3d"
name="identifier"/>
    <meta
      content="X3D-Edit, http://www.web3d.org/x3d/content/README.X3D-
Edit.html" name="generator"/>
    <meta content="license.html" name="license"/>
  </head>
  <Scene><!--Example scene to illustrate X3D tags and attributes.--><Group>
  </Scene>
</X3D>
```

Som SVG støtter også X3D skripting mellom <Script> </Script> merkene. XSLT kan brukes for å transformere X3D-dokumenter til andre formater som for eksempel VRML97 eller XHTML og SVG, selv om en SVG transformasjon ennå ikke er skrevet.
Heller eksisterer ikke så vidt meg bekjent XSLT dokument for å konvertere SVG til X3D. Det skal i teorien være mulig, men er et komplekst stykke arbeid, og man må subjektivt bestemme hvordan dette skal utføres. Det vil gjerne være gitt av intensjonen til X3D dokumentet.
For å lage X3D dokumenter i dag benyttes ofte egne applikasjoner eller applikasjonsgrensesnitt.
Eksempelvis kan nevnes Xj3D, et Javaverktøy og en X3D-dokumentleser for å lage X3D-dokumenter som samsvarer med X3D-standarden. Xj3D brukes ofte til utvikle prototyper som er ment å være utvidelser og nye egenskaper ved standarden.

Det eksisterer også vevapplikasjoner som omformer fra et dokument- eller dataformat til X3D formatet. Et eksempel er bruk av skriptspråk i CGI-applikasjoner for nedlasting og X3D-koding av GIS data fra en database eller andre dokumenterkilder. Skripting kan utføres i flere mulige skriptspråk som PHP og Python for å nevne noen eksempler.

Et X3D-dokument er på tekstformat og således lett å lage og redigere i et hvilket som helst program for tekstbehandling. Det er utviklet egne X3D-tekstbehandlere som har innebygd funksjonalitet spesielt tilpasset X3D-koding. Typisk funksjonalitet er syntaksmerking, X3D validering (sikre at X3D-dokumentet er syntaktisk korrekt). Ytterligere funksjonalitet ville vært en innebygd X3D-fortolker for å visualisere dokumentet mens man lager X3D-dokumentet. X3D-Edit er et slikt kjent verktøy (se http://www.web3d.org/x3d/content/README.X3D-Edit.html).

Det finnes ikke så fryktelig mange verktøy til å fortolke og visualisere X3D-dokumenter. Det er ikke mange åpne kildekodeløsninger. Det finnes avspillingsapplikasjoner for X3D-dokumenter som er gratis å laste ned til hjemmebruk.

Et eksempel på åpen kildekodeløsning er Flux Web3D Engine. Et kommersielt eksempel er Octaga spilleren som er gratis for hjemmebruk. Selv om Octaga er et kommersielt produkt, syntes jeg det er artig å nevne dette fordi det er norsk produkt som har høstet mange lovord internasjonalt. Les mer på deres hjemmeside http://octaga.no/.

Felles er at de leverer programtillegg for visualisering av X3D-dokumenter i en nettleser på flere OS plattformer. Det er ikke utviklet mange slike programtillegg i skrivende stund.

Bruk av programtillegg i nettleser støtter opp om mitt rammeverk for visualisering av nedbørdata i 3D. I forbindelse med visualisering av SVG dokumenter påpekte jeg fordelene og prinsippet med bruk av programtillegg. Nå er kanskje argumentet for bruk av programtillegg ytterligere underbygget?

8.8 Bearbeiding av nedlastede nedbørdata av bruker

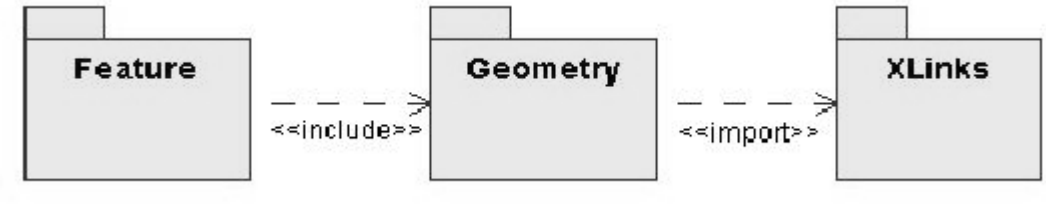
I visualiseringen av kartdokumentet kunne man tenke seg et interaktivt grafikkverktøy for nedlasting av geospatiale data som nedbørdata til bruker. Bruker vil ha muligheten til selv å bearbeide eller på annen måte benytte dataene i andre sammenhenger. Ved å implementere WFS inn i rammeverket er det mulig å uveksle data på GML (Geographic Markup Language) format.
GML er et modelleringsspråk og XML-markeringsspråk for geografiske systemer og en åpen standard for utveksling av geografisk informasjon. GML inneholder geografisk informasjon og attributter på tekstformat som kan tolkes og analyseres av eksterne systemer.
Tolking og analysering av GML dokumenter ligger utenfor rammeverket, men jeg syntes at en drøfting av bearbeiding av GML dokumenter er relevant for denne oppgaven. Det er viktig å skille mellom geografiske data formatert på GML format og videre fortolkning eller visualisering, som for eksempel et kartobjekt. Geografiske data representerer verden i romlige termer, og er ikke fokusert på fortolkningen av denne informasjonen som for eksempel en grafisk representasjon. Det er viktig å ha denne distinksjonen klart for seg.

Når det er sagt, så er det mulig å visualisere GML dokumenter grafisk.

Vi skal først studere overordnet GML dokumentstruktur. GML består av tre grunnleggende XML-skjema dokumenter. Skjemaene definerer abstrakte typer som må implementeres i

GML Applikasjonsskjemaer [se, http://www.ia.hiof.no/~gunnarmi/geomatikk/GML-Schema.ppt].

Figur 72: XML Skjema [http://www.ia.hiof.no/~gunnarmi/geomatikk/GML-Schema.ppt]



- **FEATURE.XSD** Definerer den generelle tema-modellen.
- **GEOMETRY.XSD** Inkluderer detaljerte geometri-spesifikasjoner.
- **XLINKS.XSD** Inneholder XLink-attributter som benyttes for å definere sammenhengen mellom objekter i skjemaet. Benyttes også for å definere eksterne ressurser med bruk av URL.

De tre GML skjemaene er ikke tilstrekkelig for lage kartobjekt-instanser i XML.
XML Applikasjonsskjema deklarerer kartobjekt typer og deres egenskaper ved å ”arve” fra de grunnleggende skjemaene.

Eksempel på GML dokumentstruktur:

```
<?xml version="1.0" encoding="windows-1252" ?>
<schema targetNamespace="http://www.opengis.net/examples"
      elementFormDefault="qualified" version="2.03"
                    xmlns="http://www.w3.org/2001/XMLSchema"
      xmlns:gl="http://web.gisline.no/gml"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      xmlns:gml="http://www.opengis.net/gml">
     <annotation>
          <appinfo />
          <documentation xml:lang="en">GML Skjema
nedbørdata</documentation>
     </annotation>

     <import namespace=http://www.opengis.net/gml
                  schemaLocation="feature.xsd"/>
```

`Feature.xsd` må importeres for å benytte grunnleggende typer i GML

Eksempel: GML skjemaer for nedbørdata:

```
<element name="Nedbør" type="gl:NedbørSkjema"
                  substitutionGroup="gml:_FeatureCollection" />
<element name="mengde" type="ex:MålenhetSkjema"
                  substitutionGroup="gml:_Feature" />
<element name="Nedbør.mengde" type="ex:Nedbør.MåleenhetSkjema"
                  substitutionGroup="gml:featureMember" />
```

`substitutionGroup` skal benyttes for alle klasser (feature og geometry) og egenskaper som defineres.

En samleklasse bruker `featureMember` egenskapen for å deklarere hvilke typer den kan inneholde.

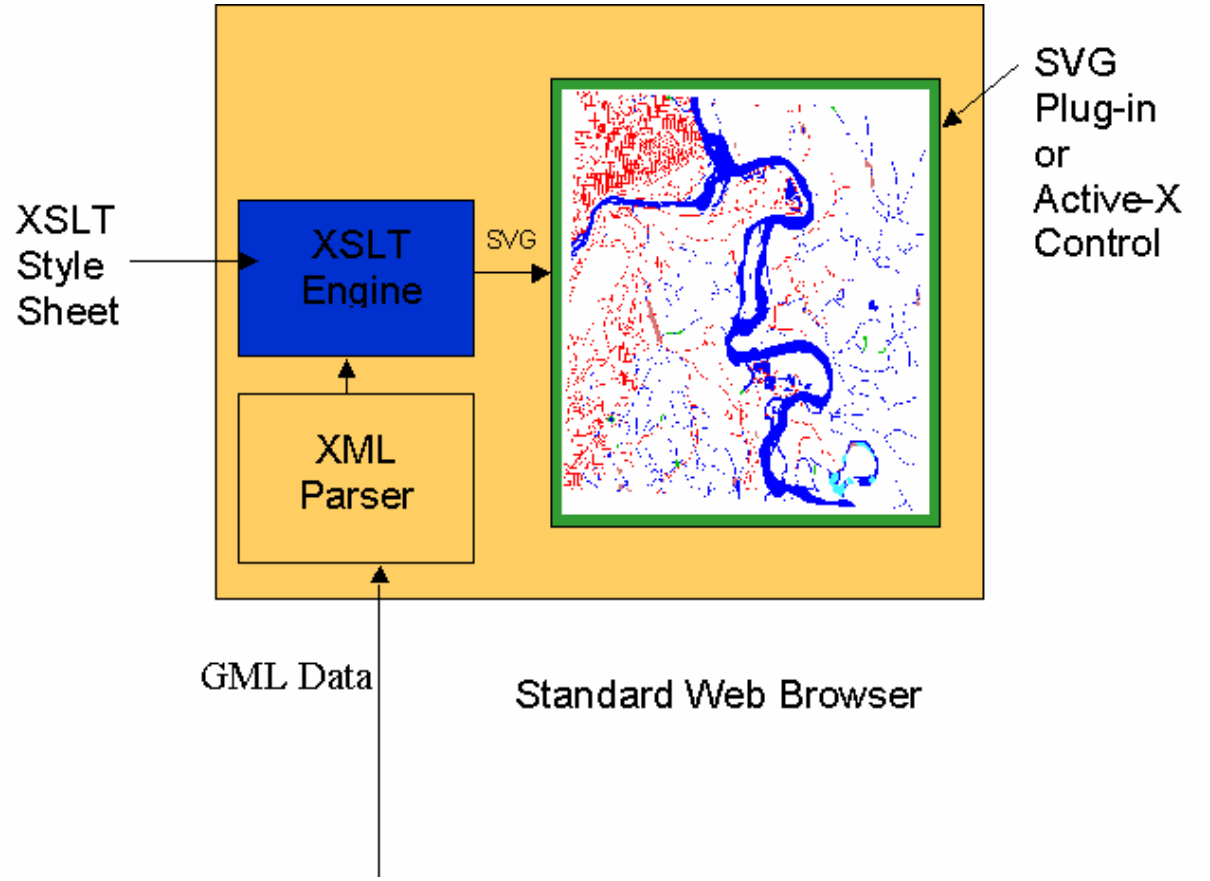
Eksempel MåleenhetSkjema:

```
<simpleType name=" MåleenhetSkjema ">
          <restriction base="string">
               <enumeration value="mil" />
               <enumeration value="meter" />
            .
            .
               <enumeration value="mm" />
          </restriction>
     </simpleType>

<simpleType name="TallmengdeSkjema">
          <restriction base="integer">
               <enumeration value="0" />
               <enumeration value="50" />
            .
            .
                <enumeration value="1500" />
            .
            .
         </restriction>
     </simpleType>

   <Nedbør.mengde>

       <mengde>
           <tallstørrelse>100</ tallstørrelse >
           <enhet>mm</ enhet >
           <areal>
               <gml:LineString
   srsName="http://www.opengis.net/gml/srs/epsg.xml#32633">
                   <gml:coordinates>
   249445.630,6649426.139 249434.179,6649431.993 249420.971,6649438.562
   .
   .

                  </gml:coordinates>
              </gml:LineString>
          </ mengde >
      </ mengde >

   </ Nedbør.mengde >
```

Hvordan visualisere nedbørdata grafisk med utgangspunkt i et GML dokument?

Siden GML er et XML-språk, så kan vi benytte XSLT for å transformere et GML dokument til SVG, VRML eller X3D. Dokumenter formatert i disse formatene kan visualiseres i en nettleser med bruk av programtillegg eller innebygd støtte i nettleseren. Det finnes også andre programmer som kan tolke og visualisere disse formatene.

Figur 73: GML - SVG transformasjon [http://www.w3.org/Mobile/posdep/GMLIntroduction.html]



Google Earth er en populær applikasjon som kombinerer Google søkemotor med satellittbilder, kartobjekter, terreng og 3D bygninger til å visualisere geografisk informasjon. Google Earth leser dokumenter basert på KML (Keyhole Markup Language) for å vise frem ulike lag. Disse dokumentene definerer hva og hvilke bilder som skal vises. Forskjellen mellom GML og KML er formingsmotoren for KML som nettopp er Google Earth. Google Earth kan sammenliknes med programtillegget ASV.

Å lage en formingsmotor basert på GML er faktisk ikke en komplisert affære.
Se Figur 73 som skisserer prosessen.

Installer f. eks SAXON (en XSLT motor) som en servlet. Det neste er å lage et stilark for å transformere GML til SVG. Vi benytter programtillegget ASV i nettleseren for å visualisere dokumenter formatert på SVG format.
GML dokumentet må så konstrueres. Dette må man uansett også gjøre for Google Earth, bare at språket er KML. Neste steg er å utforme vevsider som overfører stilarkreferansen (URL) og datareferansen (URL) til GML dokumentet til servlet. Responsdokumentet er formatert på SVG og visualiseres i nettleser som beskrevet mange ganger tidligere i denne oppgaven. Applikasjonen kan utvides med bildeunderlag (punktgrafikk) referert i forespørselen, siden SVG også støtter punktgrafikk. Resultatet må kunne kalles en minivariant av Google. Se

artikkel av Ron Lake [http://geoweb.blog.com/313900/], og introduksjon til GML [http://www.w3.org/Mobile/posdep/GMLIntroduction.html]

Meteorologisk institutt har laget egne KML-dokumenter for værvarsel med mer. Se Figur 74. Ikke overraskende kan også GML transformeres til KML med bruk av XSLT.

Figur 74:Meteorologisk institutts Google Earth applikasjon som viser nedbørdata [http://met.no/kml/index.html]

9 Konklusjon

Vi har sett på hvordan vi kan lage et rammeverk basert på åpen kildekode og standarder for visualisering av nedbørdata. Selv om rammebetingelsene tilsier griddede nedbørdata, er dette ikke en betingelse for bruk av mitt rammeverk. Tydelig kan vi se andre bruksområder enn visualisering av kun nedbørdata. Rammeverket støtter andre klimadata via verdensveven og i en nettleser. Det er heller ikke gitt at det må være klimadata. Faktum er at rammeverket støtter alle typer temporale geospatiale data.

Oppgaven inkluderer drøftelser om kunnskap som jeg mener bør ligge til grunn for å utvikle et tilsvarende rammeverk eller bruk av rammeverket for å visualisere andre typer data.

Bruk av åpne standarder sikrer konforme løsninger for visualisering og utveksling av geospatiale data. GIS-visualiseringer vil bli mer og mer benyttet i fremtiden og nedbørdata vil kun være et aspekt ved å visualisere stedfestet informasjon.

Vi har drøftet ulike metoder for datamodellering og visualisering. Vi har i den forbindelsen drøftet transformasjon av raster- og vektorgrafikk, geometrisk modellering, datautvelgelse samt visualisering med bruk av XML.

Det jeg har erfart, er at det er nyttig og formålstjenlig å bruke åpne standarder, og jeg håper oppgaven belyser dette på en grei måte. I denne oppgaven har jeg spesielt hatt fokus på åpne grafiske standarder for å visualisere primært 2D grafikk, men også 3D grafikk i en nettleser. Strategien i mine øyne er konsekvent bruk av programtillegg når det finnes løsninger for dette. Jeg stiller meg kritisk til bruk av subimplementeringer av standarder. Rammeverket støtter begge tilnærminger til problemstillingen. Jeg vil uansett understreke min holdning til denne problematikken, som går ut på å unngå plattformer med ”halvgjorte” implementeringer, hvis mulig.

Bruk av programtillegg reduserer arbeidsmengden og gir stabile og konforme løsninger for visualisering hos bruker.

Interaktive og dynamiske løsninger er både attraktive og nødvendige for visualisering av nedbørdata og andre klimadata. SVG og X3D støtter animasjon, multimedia og andre grafiske standarder, samt bruk av ECMA-skript. Ekstra responsive vevsider får man med å bruke AJAX-metodikken.

Jeg har drøftet rammeverket ved bruk av griddede nedbørdata og tradisjonell visualisering, som en rød tråd gjennom oppgaven. Det burde ikke være til hinder for å se at rammeverket kan benyttes til å lage andre typer visualiseringer.

Visualisering i 3D er mulig i nettlesere. Støtten til 3D visualisering er ikke like gjennomført som for 2D, men den er på veg. Bruk av 2D eller 3D må vurderes som all annen visualisering, nemlig med fokus på ”budskapet” og brukervennlighet.

Vi har vist at rammeverket støtter andre standarder gjennom transformasjoner med bruk av CGI-skript og XSLT. Eksplisitt transformering med bruk av spesialprogrammer er også et alternativ.

Å transformere data fra 2D til 3D følger samme metodikk. Å konvertere SVG til X3D er teoretisk mulig, men det eksisterer i dag ikke verktøy for dette som fullt ut støtter standardene.

Det eksisterer i dag spesialløsninger for konvertering med fokus på temaet for visualiseringen. Jeg tror flere slike løsninger vil bli mer vanlig enn å utvikle XSLT-dokumenter for å transformere SVG til X3D.

9.1 Praktisk gjennomføring

Her følger en kort redegjørelse av mitt arbeidsmiljø og maskinvareplattform. Jeg har benyttet en Linux-tjener med "Debian Sarge" (se http://www.debian.org/) distribusjonen installert. Dette er en veldig stabil installasjon, men det går på bekostning av tilgang til nyeste versjoner av programvare.
Der det har vært tvingende nødvendig for meg å ha en oppdatert versjon av programvaren, har dette blitt løst ved å laste ned kildekoden og kompilere og installere programvaren selv.

Grunnet brannmur og generelle sikkerhetsregler ved instituttet, har denne maskinen ikke vært tilgjengelig via Internett, kun intranett. Hensikten med å jobbe i mitt restriktive nettmiljø, er for å ha tilgang til nødvendig programvare og klimadata. Nær tilgang til klimadata har vært en klar fordel for gjennomføringen av den praktiske delen av oppgaven.

Mitt arbeid har blitt og er tilgjengelig via egen vevtjener med utlegg av eksempler på bruk av rammeverktøyet og programvare for andre instituttansatte, spesielt GIS gruppemedarbeidere.

Jeg har benyttet min Windows arbeidsstasjon som har fungert som klientmaskin under testkjøring. Jeg har også brukt tjener som klientmaskin for å teste ut nettlesere for Linuxplattformen.

Rammeverket skal kunne implementeres på de fleste plattformer Linux, Unix og Windows.

En "alternativ" måte å installere rammeverket på, er "LiveDistro", som er et generelt begrep for kjøre et operativsystem og programvare som er installert på CD, DVD, USB eller flashenhet. Målet med å skille kildedata (nedbørdata i dette tilfellet) ut av rammeverket syntes da å være klar. Jeg selv ikke laget en slik distribusjon, derimotjeg har verifisert at det eksisterer slike distribusjoner. Et eksempel er "LiveCD" distribusjonen GEOLivre Linux (se, http://distrowatch.com/?newsid=03183).

9.2 Videre arbeid med rammeverket

Rammeverket, slik det fremstår i dag, er allerede ferdigutviklet til å "møte" fremtiden. Å bygge et rammeverk basert på viktige og åpne standarder for dataoverføring og grafikk sikrer rammeverket for fremtiden. Bare den stunden jeg har arbeidet med oppgaven har det skjedd flere programvareendringer. Åpne standarder finner veien inn i kommersielle løsninger. ESRI jobber blant annet med å implementere programmoduler for å fortolke SVG-standarden.

Slik det ser ut i dag så vil SVG og X3D være de viktigste standardene for hhv. 2D og 3D grafikk. I fremtiden vil det bli utviklet mer og mer 3D løsninger. Det betyr ikke at 2D blir "udatert", men en nærliggende tanke er at 2D og 3D integreres i en standard.

Åpen kildekode som jeg har benyttet meg av, er også i en rivende utvikling.
GRASS manglet for eksempel rutiner for å omforme data fra 2D til 3D uten å gå omveier via en rekke applikasjoner, noe som har vært situasjonen til nå. Suksessen til GRASS i mine øyne må tilskrives den modulære oppbygningen, som gjør av mange utviklere kan delta i videreutviklingen av GRASS uten å måtte forholde seg til en kompleks og uoversiktlig programvarestruktur.
PostGIS har implementert funksjoner for å returnere data formatert i både SVG og GML. PostGIS er mer enn å formatere romlig data på vektorformat – det er over 300 funksjoner som er laget for håndtering av romlige data. Listen med funksjoner blir stadig lenger.

Fremtidig arbeid med rammeverket vil i mine øyne være bygge på de samme komponentene som i dag, dvs. oppgradere til nyere versjoner for å utvide og forenkle eksisterende rutiner. Et naturlig skritt videre, vil være å lage flere og alternative 2D og 3D løsninger for visualisering av klimadata (ikke kun nedbørdata).

Et mål for fremtiden - sett fra et lokalt perspektiv - vil være å integrere arbeidet med nedbørdata og andre klimadata for å få personer til å bruke rammeverktøyet. Det vil gi en mer helhetlig GIS struktur ved at eksisterende rutiner for overføring og konvertering av nedbørdata i rammeverket kan droppes. En åpenbar fordel er også at flere fagpersoner og fagmiljøer blir brakt sammen, noe som bidrar til å heve kompetansen og gir en mer helhetlig forståelse for håndtering av værdata.
Rammeverket vil i fremtiden strebe etter å støtte andre standarder og formater for utveksling av data.
Mange land sliter økonomisk, og dette gjenspeiler seg i budsjettene til de enkelte nasjoners meteorologiske institutter. Ved å få personer til å bruke rammeverket kan man utveksle data og kompetanse med veldig lave kostnader. Rent økonomisk trenger man ikke kun å se på land med dårlig økonomi. Faktum er at effektiviseringsprosesser er i "vinden", og omstilling og kostnadskutt er ord man nesten hører daglig.
Jeg har stor tro på at rammeverket mitt av disse grunner vil "selge" seg selv – det kan gå et par år for å personer å til å jobbe med rammeverket, men omstilling, opplæring og endring av rutiner, er prosesser som tar litt tid.

Bibliografi

(Teknologiforum), R. (2006). "Rammeverksdokument – Norge digitalt." Rammeverk og infrastruktur for stedfestet informasjon i Norge Retrieved 14.02.2006, 2007, from http://www.statkart.no/IPS/tinglysing/filestore/Norge_Digitalt/Teknologi/rammeverk/Rammeverksdokument_20061006.pdf.

Albregtsen, F. and G. Skagestein (2006). Digital representasjon av tekster, tall, former, lyd og video. Oslo.

Amlien, J., T. Bøhn, et al. (1992, 01.01.1997). "Innføring i digital bearbeiding av satellittbilder, digital kartbehandling og geografiske informasjonssystemer." 2006, from http://www.geo.uio.no/geogr/geomatikk/prosjekt/forordg.html.

Bjørke, J. T. (2005). DATAMODELLER, ALGORITMER OG DATASTRUKTURER FOR GEOGRAFISK INFORMASJONSBEHANDLING, Universitetet for miljø- og biovitenskap.

Bjørke, J. T. (2005). Digitale terrengmodeller, Universitetet for miljø- og biovitenskap.

Bjørke, J. T. (2005). Kartografisk kommunikasjon.

Bourke, P. (1989). "Triangulate Efficient Triangulation Algorithm Suitable for Terrain Modelling." 2006, from http://local.wasp.uwa.edu.au/~pbourke/papers/triangulate/index.html.

Dassau, O., S. Holl, et al. (2005). "An introduction to the practical use of the Free Geographical Information System GRASS 6.0." 2006, from http://www.gdf-hannover.de/dl.php?download=gdf_grass60_v1.2_en.pdf.

Evenden, G. I. (1990). Cartographic Projection Procedures for the UNIX
Environment—A User's Manual, UNITED STATES DEPARTMENT OF THE INTERIOR
GEOLOGICAL SURVEY**:** 68.

Geodesi, S. k. (2004, 27. mai 2004). "KOORDINATBASERT REFERANSESYSTEM." Versjon 2.0. 2006, from http://www.statkart.no/filestore/Standardisering/KRS-utgave-2004-05-27--1.pdf.

George, P.-L. and H. Borouchaki (1998). Delaunay Triangulation and Meshing: Application to Finite Elements. Paris, Hermes.

Geroimenko, V. and C. Chen (2005). Visualizing Information Using SVG and X3D. London, Springer.

Gjevestad, J. G. (2006). En introduksjon til referansesystemer og referanserammer. GeoForum. Kristiansand, Universitetet for Miljø- og Biovitenskap. 2007.

Group, S. W. (2007). "Scalable Vector Graphics (SVG)
XML Graphics for the Web." 2006, from http://www.w3.org/Graphics/SVG/.

Group, T. P. G. D. (2005). "PostgreSQL 8.0.7 Documentation." from http://www.postgresql.org/docs/8.0/interactive/index.html.

Gudgin, M., M. Hadley, et al. (2003, 24 June 2003). "SOAP Version 1.2 Part 1: Messaging Framework." from http://www.w3.org/TR/soap12-part1/.

Hafskjold, C. (2004). Offentlige strategier for åpen programvare – dagens situasjon og Teknologirådets anbefalinger. AFIN-seminar. Avdeling for forvaltningsinformatikk (AFIN) Teknologirådet.

Heywood, I., S. Cornelius, et al. (2006). An Introduction to Geographical Information Systems, Prentice Hall.

Inc, R. R. (2006). "PostGIS Manual." 2006, from http://www.postgis.org/docs/.

kartverk, S. (2004). KOORDINATBASERT REFERANSESYSTEM. DATUM, KOORDINATSYSTEM,TRANSFORMASJON, KONVERTERING OG AVBILDNING. Hønefoss, Statens kartverk**:** 42.
Lake, R., D. Burggraf, et al. (2004). Geography Mark-Up Language: Foundation for the Geo-Web, Wiley.
Longley, P. A., M. F. Goodchild, et al. (2005). Geographic Information Systems and Science, Wiley.
Lundestad, E. (2004). Rekonstruksjon av klimaet på Hamar 1749 – 1835 basert på gårdsdagbøker. Hovedoppgave ved Institutt for geografi. Institutt for geografi. Bergen, Universitetet i Bergen. Cand.scient.: 283.
Moe, M. (1995). Klibas - the DNMI climatological database system. Oslo.
Mortenson, M. E. (2006). Geometric Modeling. New York, Industrial Press Inc.
Refnes, H. "Introduction to SVG." 2006, from http://www.w3schools.com/svg/svg_intro.asp.
Selinger, P. (2003). "Potrace: a polygon-based tracing algorithm." from http://potrace.sourceforge.net/potrace.pdf.
Thorpe, W. (1998, Thu Jun 4 15:27:43 1998). "A GUIDE TO THE WMO CODE FORM FM 94 BUFR." from http://dss.ucar.edu/docs/formats/bufr/bufr.pdf.
Tollan, A. (1993). "NORDENS HYDROLOGI." Retrieved 5.12, 2006, from http://www.uio.no/studier/emner/matnat/geofag/HYD2010/v06/kompendium_hyd1/nordens_hydrologi.pdf.
Vretanos, P. A. (2002, 17-MAY-2002). "Web Feature Service Implementation Specification." from https://portal.opengeospatial.org/files/?artifact_id=7176.
Web3D Consortium, I. (2004, Nov 2005). "ISO/IEC 19775:2004." Extensible 3D (X3D), 2007, from http://www.web3d.org/x3d/specifications/.
Webster, R. and M. A. Oliver (2001). Geostatistics for Environmental Scientists. Ontario, John Wiley & Sons Canada, Ltd